A Software System for Machine Translation

Jun-ichi NAKAMURA

Doctoral Thesis

Kyoto University

1988

October

A Software System for Machine Translation

Jun-ichi NAKAMURA

Doctoral Thesis

Kyoto University

1988

October

A Software System for Machine Translation

Jun-ichi NAKAMURA

Abstract

In this thesis, a software system for a practical machine translation system is described. This system was developed for the Mu machine translation project, which was supported by the Science & Technology Agency of the Japanese Government. The Mu project began in the fall of 1982 and ended in the spring of 1986. The basic designing of this software system started in 1981.

For practical machine translation (MT), a software system for its development and maintenance is crucial, because we need to handle large amounts of complex linguistic information. Since the linguistic information is expressed with grammars and dictionaries, we need a software system which consists of sub-components: a grammar writing language to describe complex linguistic phenomena; and a dictionary database to enable the handling of large word specific information. In addition to these sub-components, we also need software to support the development and maintenance of grammars and dictionaries.

In this thesis, first we discuss a software environment for a practical MT system to clarify the requirements of a software system. Then the grammar writing language called GRADE is described, which has been developed to fulfil the requirement of a grammar writing language. GRADE has the features: (1) rewriting rules based on tree-to-tree transformation with flexible pattern matching functions; (2) control mechanism with subgrammars and subgrammar networks; (3) dictionary rules to express word specific linguistic phenomena.

To demonstrate the usefulness of GRADE, we discuss typical problems in grammars for an MT system; such as lexical and structural ambiguities in English sentence analysis, word selection and structural adjustment in transfer from Japanese into English. After that, their solutions, using GRADE, are explained. We also discuss the outline of the grammars for a Japanese-to-English translation system, which were developed in the Mu project. This shows the way to use GRADE in a practical system.

For a dictionary database system, a 'neutral dictionary' system is proposed. This system separates dictionaries into two parts: neutral and general purpose dictionaries, which are written by lexicographers; and processing-specific dictionaries, which are transformed from the neutral dictionaries. Software tools to support this separation are also explained.

For the development and maintenance of large and complex grammars and dictionaries, software packages have been developed. The packages consists of three components: the translator and the executor for GRADE grammar rules; tools for grammar development; and tools for dictionary maintenance. These packages were practically used in the Mu project. Details of the packages are also explained.

Acknowledgments

The author would like to thank to Professor Makoto Nagao of Kyoto University for the supervision and continuous encouragement to complete this thesis.

The author also wishes to thank Professor Jun-ichi Tsujii of Kyoto University for his constructive and useful discussions during the course of this study.

His thanks are also due to Professor Satoshi Sato of Kyoto University, Dr. Yoshiyuki Sakamoto of the Electrotechnical Laboratory, Mr. Masayuki Sato of the Japan Information Center of Science and Technology, Mr. Fumito Nishino of Fujitsu Laboratories, Mr. Masami Kogi, Mr. Yoshinori Sakane and Mr. Hirohisa Kosaki of Toyo Information Systems, Mr. Masahiro Kobayashi of NEC, Dr. Yasuhiro Katagiri of NTT, and other participants of the Mu project for their contribution to this study.

He would like to thank Dr. John Bateman of University of Southern California, who was in Kyoto University, and Mr. Alen W. Black of Kyoto University for their useful comments by reading the draft of this thesis. They also pointed out errors and rhetrical problems, and showed me precise and readable alternatives.

Last but not least, he is grateful to all members of Professor Nagao's laboratory.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Aims of Machine Translation Research

Development of a machine translation (MT) system provides many interesting topics from both research (computational linguistics, and software science) and practical points of view. Some of the topics are the followings:

- MT is a good application of computational linguistics and natural language processing research. We can verify models proposed by them in large scale.

- Although we should stress the use of linguistic information in a practical system, we still need to apply the techniques developed in artificial intelligence research, such as a knowledge representation method. However, we need to find a *practical* way to use them, because the domain of the traget of a machine translation system is larger than that usually round in artificial intelligence research.

- Since a practical machine translation system is a large and complex software system, we need to develop various software to support development and maintenance of a system. This helps us find new methods in software technology.

- A machine translation system is *practically* requested by many organizations, which want to translate huge amounts of documents, such as abstracts of technical papers, operation manuals of technological products, and so on. Translation of these documents require clear and uniform translations and it is not easy for this to be achieved by human translators.

Brief History of Machine Translation Research

The research and development of a machine translation has long history [Slocum84] [Tamati85], and started at the same time of computer science [Tadenuma61]. However, the 1966, highly negative, ALPAC report [ALPAC66] has suppressed research activities of a machine translation system (at least in research organizations). This report denied practical merits of a machine translation system mainly from an *economical* point of view at that time.

Since the publication of the report, the situation around machine translation research, however, was changed. The following should be pointed out:

- Computational linguistic research was advanced [Sells85]. Basic frameworks for natural language processing became more clear than during the 1960's.
- Speed and cost of processing have been improved. Thus a system is able to do more complex operations.
- Memory size available for processing has been greatly increased.
- Size of disk storage to store linguistic information, such as dictionaries, has been increased.
- The environment to handle character data has been greatly improved. For instance, handling Japanese characters (Kanji characters) does not restrict the operation of the system. High quality computer typesetting becomes also available [Knuth86].

These improvements have removed some of the *bottlenecks* claimed in the ALPAC report. In early 1980's, several researchers, especially in Japan, re-started the research of a machine translation.[1] Groups in research organizations have started basic studies of machine translation [Nagao80a] [Nagao82a] [NishidaF80] [NishidaT82] [Iida82] [Sakaki84], and groups in commercial companies also have started the development of practical machine translation systems [Uchida82] [Muraki83] [Nitta82]. In such a situation, 'the Mu project' started in 1982.

[1] Research and development supported by private funding in U.S., and supported by governmental funding in France and Canada, for instance, continued even after the publication of the ALPAC report.

The Mu Project

The Mu project is one of the first big projects to aim to develop a *practical* machine translation system [Nagao83b]. The Mu project was a 4 year project from 1982 to 1986, and its aims were the following:

- Development of a *practical* machine translation system between Japanese and English for the translation of abstracts of scientific and technological papers. The Japanese corpus is abstracts which are published by Japan Information Center of Science and Technology (JICST), and the English corpus is abstracts in INSPEC document database.[2]

- Development of a *software environment* for a machine translation system and other natural language processing systems.

- Construction of large *dictionary databases* and a software system to handle them.

- Development of a unified software environment for users and developers (grammar writers and lexicographers) of a machine translation system.

The Mu project was performed through Special Coordination Funds for Promoting Science & Technology of the Science and Technology Agency of the Japanese Government. The study of this thesis was partly supported by this project.

Software Environment for Machine Translation Systems

A machine translation system consists of the following components:[3]

- *Grammars* for translation. A 'grammar' is usually divided into three parts: an analysis grammar, a transfer grammar, and a generation (or synthesis) grammar.[4]

[2]INSPEC document database covers fields such as electronics, computers & control etc., and is published by the Institute of Electrical Engineers in Great Britain.

[3]In this thesis, we restrict the discussion on a machine translation system which only uses sentential information, that is, a system which translates a text sentence-by-sentence. The use of context information, for instance, is a future research topic for the development of machine translation systems. More details are discussed in chapter 2.

[4]This three-step framework is called 'transfer approach.' Another framework called 'interlingua approach' is also proposed. The difference between them are discussed in [Tsujii88].

A framework for writing grammars (grammar writing language) should be powerful enough to handle various linguistic phenomena, and must satisfy the following requirements:

  - A framework to describe complex linguistic phenomena, and allow the use of heuristic information for translation.
  - A language which allows to expression of modularity of grammars, selection of most preferred output, and word specific information.
  - A general purpose grammar writing language applicable to all phases of a machine translation system.

- *Dictionaries* which store large amounts of linguistic information on words. Since the dictionary data is useful for all natural language processing researches, we should separate the dictionary system into two parts: the development of dictionary data and the utilization for processing.

- *Software packages* for the development and maintenance of the grammars and dictionaries.

Because grammars and dictionaries for a *practical* machine translation system are very large and complex, a software system to support their development is crucial.

Keeping these points in mind, the author of this thesis has developed a software system that consists of the following sub-systems:

- *Grammar writing language GRADE*, which supports:
  - Grammar rules based on tree-to-tree transformation with flexible pattern matching. This allows complex linguistic phenomena to be described, and is suitable for expressing all grammars of a machine translation system.
  - A mechanism to control the translation process, and to allow the modurality of grammars.
  - A mechanism to embed local ambiguity to select most preferred output.
  - Dictionary rules to describe word specific linguistic phenomena.

- *Neutral and general purpose dictionary system*, which separates the utilization of dictionaries from the development and improvement of dictionaries.
- *Software packages for the development and maintenance*, which consists of three sub-systems:
  - Software for the execution of grammar rules.
  - Tools for the development of grammar rules.
  - Tools for the maintenance of dictionaries.

Usefulness of this software system is demonstrated in the course of the Mu project.

Summary of Thesis

To show problems and their solutions in the development of a *practical* machine translation system, five topics about a software system are discussed in the following chapters. First, software requirements in a machine translation system are discussed in chapter 2. Based on this discussion, chapter 3 describes the grammar writing language GRADE. Then chapter 4 shows grammars written in GRADE, which have been developed in the Mu project, to demonstrate the usefulness of GRADE. Since dictionary databases are also important in a machine translation system, chapter 5 discusses the utilization method of dictionary databases. Finally, chapter 6 explains the configuration of the GRADE system in details: this includes software tools for the development and maintenance of grammars and dictionaries. In chapter 7, concluding remarks are given.

6項欠

Chapter 2

Software Environment in Machine Translation

2.1 Development of Practical Systems

A *practical* machine translation (MT) system is always in the sate of under development. We need to improve grammars to cover sentence styles typically used in target documents, and to improve dictionaries to cover target fields. To develop and maintain a practical MT system, thus, we need various software to help the improvement of the system.

In this chapter, we first consider a software environment of an MT system as a whole. Then, we discuss features of the software environment necessary for the development of a large practical MT system, which are the following sub-systems:

- Grammar writing languages for analysis, transfer, and generation.
- Software to develop complex grammars.
- Software to maintain large dictionaries.

Although various other software packages, such as user interfaces, are also necessary for practical use of an MT system, these sub-systems are the most important parts to improve the quality of an MT system.

2.2 Classification of Software Environment

A practical MT system needs five components (see also figure 2.1):

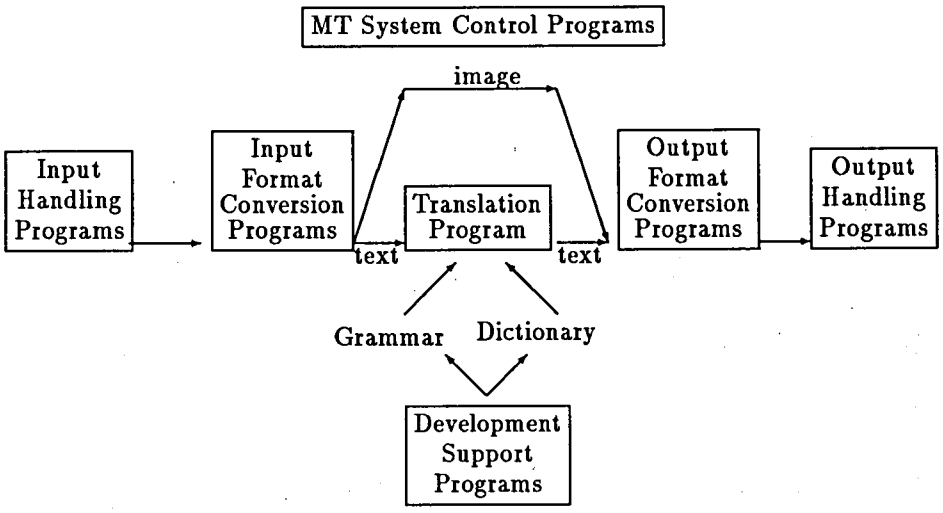


Figure 2.1: General Configuration of Software System of Machine Translation

1. Input and output handling programs.

2. Input and output format conversion programs, include pre- and post-editor.

3. A translation program, which refers to grammars and dictionaries.

4. MT development support programs.

5. MT system control programs.

In the rest of this section, each of them is discussed.

Input and Output Handling Programs

These programs work as interfaces of the MT system with source and target documents. To support various applications, the MT system should have a flexible interface with input and output programs: optical character recognition (OCR) systems, text editors, graphic printing systems, and so on. Note that this part is practically important, because the cost of inputing text and the quality of output should be better than *human* translations.

Input and Output Format Conversion Programs

Input documents usually contain not only sentences which will be translated, but also non-translatable parts, such as figures and image data. Input documents sometimes contain formatting information for computer typesetting (e.g. LaTeX [Lamport86]). Thus the format conversion program dividing input documents into sentences and other parts is also necessary for the MT system.

The output format conversion program, then, needs to combine the translated textual part and the non-translated part (see figure 2.1). This processing is not trivial, and we need to develop various sub-programs.

Programs for pre-editing and post-editing are also necessary, because the kernel of the MT is not perfect now. Note that the input conversion program may distinguish the *type* of target sentences automatically or manually (with pre-editing): titles, *ordinary* sentences, mathematical formulas, and so on. This distinction is useful for the kernel of the MT system, because the system does not need to *analyze* mathematical formulas, for instance.

Translation Program (Kernel of MT)

Referring to grammars and dictionaries, this part translates sentences. In a practical MT system, grammars and dictionaries are very large and complex, because they need to handle various linguistic phenomena. The translation program should support this complexity. Details are discussed later.

MT Development Support Programs

Software tools to develop and maintain grammars and dictionaries are also important, the same as in a large software system. An MT system however has different problems from conventional software, because the maintenance phase of an MT system is not clearly distinguished from the development phase. For example, to translate different type of documents from the originally designed one, we should modify a large portion of the grammar. To translate a different field of text, we should develop large dictionaries for the new field.

MT System Control Programs

Since a practical MT system consists of many components as discussed, software to integrate these components is also required, such as the system proposed in [Yada83].

Although programs to convert input and output data (1, 2) are important in a practical system, we focus on software systems to develop grammars and dictionaries (3, 4), because these are the most critical parts of an MT system. In the rest of this chapter, we discuss the following topics to clarify the requirement for a practical MT system:

- Grammar writing languages.
- Software for the developemnt of grammars.
- Software for the maintenance of dictionaries.

2.3 Grammar Writing Language

Since grammars used in a practical MT system become more complex and precise in proportion to the system development, it is impractical to express them in an ordinary programming language. Furthermore, grammars are continuously modified to handle *new* linguistic phenomena which will be found during the system development and practical use. We need thus to design a special *grammar writing language* (a task-oriented programming language) first, then we start the development of grammars using this special language.

The translation process of an MT system is usually divided into three phases:[1]

- *Analysis phase* of an input sentence, which generates an internal structure expresses the *meaning* of the input sentence.
- *Transfer phase*, which maps the internal structure expressed in a source language into the internal structure in a target language.

[1] As mentioned in chapter 1, this three-step framework is called 'transfer approach.' Another framework called 'interlingua approach,' which does not have a transfer phase, is also proposed. [Tsujii88] discusses their difference.

- *Generation phase*, which makes a surface sentence from the internal structure.

Each phase requires different features for its expression of the grammars. In the following sections, we compare grammar writing languages for each phase.

2.3.1 Grammar Writing Language for Analysis Phase

The analysis phase is the most *atractive* part in natural language processing in general. Many researchers have developed various analysis models (grammar writing languages) until now, such as:

1. Augmented Transition Network (ATN) Grammar
2. Augmented Context Free Grammar (ACFG)
3. Tree-to-Tree Transducer
4. Others

Typical grammar writing languages based on each framework are listed in table 2.1.[2] Since the explanation of the full details of each grammar writing system is beyond the scope of this section, we compare these grammar writing languages from a view point of the development and maintenance of a large scale analysis grammar as follows:

- Handling *complex* linguistic phenomena.
- Using linguistic and non-linguistic *heuristic* information.
- Supporting the development of a complex grammar, such as the function to divide a whole grammar into several *modules*.
- Choosing the most *preferred* output.
- Using *word specific* information.

[2]Details of these languages are described in references shown in table 2.1, [Nikkei83], and [Yoshida85].

Table 2.1: List of Typical Grammar Writing Systems

Type	Language	Develop	Host Lang.	Features	Reference
ATN		Woods	Lisp	Extension of a push-down automata.	[Woods70]
		Toshiba	C	Extending the control structure of Woods' ATN.	[Amano82]
ACFG	LINGOL	Pratt	Lisp	Analysis algorithm unified top-down and bottom-up parsing.	[Pratt73]
	E-LINGOL	ETL	Lisp	Extending LINGOL to prediction of analysis.	[ETL78]
	DCG	Pereira	Prolog	Rules are directly converted into Prolog program.	[Pereira80]
	BUP	ETL	Prolog	Using DCG expression by bottom-up parsing.	[Matsumoto82]
	SAX	ICOT	Prolog	Parallel version of DCG paser for sequential machine.	[Matsumoto86]
	PATR-II	SRI	Lisp	Analysis using Unification of acyclic graphs.	[Shieber84]
Tree	System-Q	Montreal	Fortran	Using rewriting rules for directed graphs.	[Colmerauer70]
	PLATON	Kyoto	Lisp	Tree-based pattern matching with ATN-like control.	[Nagao76]
	ROBRA	Grenoble	Assembler	Controlling by subgrammar.	[Boitet79]
	GRADE	Kyoto	Lisp	Using dictionary rules.	[Nakamura83]
Others		NEC	C	Extension of a push-down automata.	[Muraki84b]
	ESPER	Fujitsu	SPL	Adding the function of exchange and deletion to ACFG.	[Uchida84]

Handling Complex Linguistic Phenomena

An MT system needs to analyze sentences in various languages: English, Japanese, Chinese, and so on, which have different linguistic phenomena. For example, the basic word ordering of Japanese is completely different from that of English. A grammar writing language thus needs to have a function to handle various linguistic phenomena.

In addition to this, target sentences of a practical MT system are more complex than *sample* sentences discussed in a model system. They contain many coordinate phrases, omissions of syntactic components, mathematical formulas, abbreviated expressions, and so on. These characteristics of sentences have sometimes context sensitivity (within a sentence). A grammar writing language therefore needs to manipulate these complexities of a language.

Augmented Transition Network (ATN) grammars [Woods70] and *Augmented Context Free Grammars* (ACFG), such as DCG [Pereira80], are extention of push-down automata and context free grammar formalism respectively. For example, grammar rules and dictionary items in DCG[3] is almost same as ones of CFG as shown in figure 2.2. DCG expresses the extension as an argument of non-terminal symbols (e.g. `sentence(s(NP,VP))`), and an auxiliary term to call Prolog predicates (e.g. `is_name(W)`), which is an implementation programming language of DCG.

Since both ATNs and ACFG are based on type 2 grammars of the formal language theory, they do not have a power to *directly* handle complex linguistic phenomena, such as context sensitivity, and they use *augmentation* to express them. Although they *can* handle context sensitivity in a theoretical sense, their lack of direct expressive power is a basic problem for the development of a practical MT system. The grammar languages of [Muraki84b] and ESPER [Uchida84] have the same problem, because they are also extensions of the type 2 grammar (CFG) formalism.

On the other hand, *Tree-to-Tree Transducers*, such as ROBRA [Boitet79], solve this problem by using the type 0 grammar formalism to express grammar rules. It can find any syntactic constituent in any place in a tree, and can move it to any place. To

[3]BUP [Matsumoto82] and SAX [Matsumoto86] in table 2.1 use the same grammar writing formalism as that of DCG. They use different parsing algorithms.

```
sentence(s(NP,VP)) --> noun_phrase(N,NP),verb_phrase(N,VP).
noun_phrase(singular,np(Name)) --> name(Name).
verb_phrase(N,vp(IV)) --> intrans_verb(N,IV).

name(name(W)) --> [W],{is_name(W)}.
intrans_verb(N,iv(Root)) --> [W],{is_intrans(W,N,Root)}.
```

(a) Grammar Rules

```
is_name(mary).
is_intrans(lives,singular,live).
```

(b) Dictionary Items

Figure 2.2: Grammar Rules and Dictionary Items in DCG

express complex linguistic phenomena in a practical MT system, we need such power of a grammar writing language based on the tree-to-tree transducer formalism.

Using Linguistic and Non-linguistic Heuristic Information

Even if we include grammar rules based on a linguistic theory into a grammar, we still need to use various linguistic and non-linguistic *heuristics*, because there are many exceptional structures in target sentences of a practical MT system. To use various heuristic information, the data structure manipulated with a grammar writing system allows encoding of various levels of information into one structure.

PATR-II [Shieber84], which is an ACFG parser based on unification grammar, tries to express such information in one structure. As shown in figure 2.3, PATR-II allows the encoding of various information into one node as a feature list:

> A VP node has a feature agr (agreement), and sub-features number and person.

Although the features in this example are syntactic ones, we can actually encode any information into this structure.

In a tree-to-tree transducer system, *annotation* of a tree structure may encode the same information as shown in figure 2.4. ROBRA, for example, allows the annotation

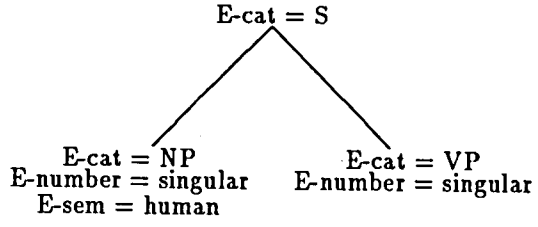
```
(1) S -> NP VP
        <VP agr> = <NP agr>
(2) VP -> V NP
        <VP agr> = <V agr>
```

(a) Grammar Rules

```
Uther:    <cat> = np
          <agr number> = singular
          <agr person> = third
Arthur:   <cat> = np
          <agr number> = singular
          <agr person> = third
knights:  <cat> = v
          <agr number> = singular
          <agr person> = third
```

(b) Dictionary Items

Figure 2.3: Grammar Rules and Dictionary Items in PATR-II



E-cat : English Category Symbol
E-number : English Number (singular or plural)
E-sem : English Semantic Marker

Figure 2.4: Annotated Tree Structure

of a tree structure with features, although the number of features are restricted. In addition, we can express any structured information (not only a syntactic structure, but also a semantic structure) in one tree, because the tree-to-tree transducer does not distinguish them. Although grammar writers need to be aware which sub-tree expresses what kind of information, this is useful when developing a practical system.

Supporting Development of Complex Grammars

Grammars for a practical MT system cannot be developed by one person, because they are very large and complex. Many grammar writers thus *cooperatively* develop grammars. This situation is different from the development of a model system, and is closer to the development of a large software system. To support cooperative development, *modularity* of the grammar is as important as that of a software system. Note that modularity of the grammar is also useful for expressing *natural steps* of the processing.

An ACFG framework does not have a function to express the modularity of a grammar, because each grammar rule is treated equally, and this *uniformness* of the grammar is a good point of the framework. However the uniformness usually causes problems in the development of a large system. Although grammar writers may introduce the modularity into their grammar rules even in ACFG-like frameworks, a grammar writing language itself should force them to make the grammar modular.

Choosing Most Preferred Output

It is difficult to resolve all ambiguities in natural sentences with current techniques. A system should choose the most *preferred* interpretation from many possibilities. Before discussing a function to choose the most preferred output, let us classify MT systems into two types:

- *Interactive MT*: This type of MT systems allows a user to interact its translation (usually, sentence-by-sentence) to choose correct analysis. This type of MT is again classified into two sub-types:
  - HAMT (Human aided MT), in which the MT system outputs translations and a user helps the translation process.

- MAHT (Machine aided human translation), in which a user translates sentences, and the system helps his/her translation work [Melby83].

This is useful for small scale applications.

- *Batch-mode MT*: This type of MT system translates many documents at once, and does not allow a user to interact its process. This system is useful, when a user needs to translate large text database, for instance.

The batch-mode MT system obviously should output a *single* translation which is most *preferred* of several alternatives. Since the selection of correct translation is not easy task for a user, even the interactive MT system also needs to choose few possible translations. To choose *preferred* translations from many alternatives, a grammar writing language should have a mechanism to ordering possible translations. This is also difficult in ACFG-like systems because of their uniformness.

Using Word Specific Information

There are many word specific phenomena in natural languages. Some words have special usages, and others have special meanings. A grammar writing language should allow the expression of such word specific information in *lexical rules*. Most grammar writing systems listed in table 2.1 do not have explicit functions to describe lexical rules.

In the unification-based systems such as PATR-II, for example, some word specific information may be expressed as a feature list as shown in figure 2.3. A feature list however does not allow the expression of *syntactic* rules directly, but it is interpreted by the grammar.

2.3.2 Language for Transfer Phase

The transfer phase maps the internal structure of a source language generated from the analysis phase into the internal structure of a target language for the generation phase. We need the following processes in the transfer phase [Nagao85c] [Nagao86a]:

- *Word selection*: The most important task in the transfer phase is to choose an adequate word for a translation. To choose a translated word, a system needs to

check various conditions. A translated verb, for example, depends on the object which is governed by the verb. We need a mechanism to express such structural conditions.

- *Structural transfer*: When the structural difference between a source language and a target language is large, the transfer phase should use a powerfull *structure-to-structure* transformation framework. Since the structure of Japanese differ from that of English, for example, we need a complex transfer between them.[4]

- *Syntactic and semantic normalization*: There are many 'concepts' which are expressed in a *single* word in one language, but are expressed in *several* words in another language, such as compound words, phrases, and clauses. For example, the English word '*meaningless*' is sometimes expressed with the Japanese phrase 'IMI NO NAI' (*'which does not have meaning'*). It is better to normalize this kind of 'redundant' expressions at the transfer phase, when we want to get a good translation. The framework to express the transfer phase should allow this processing.

To realize these processes, a grammar writing language for the transfer phase also should have the same features as discussed in section 2.3.1. Among them, the function to use *word specific* information is important, because most rules in the transfer phase are related to specific words.

2.3.3 Language for Generation Phase

A generation phase in a practical MT system is not simply a *reverse* processing of the analysis phase. The analysis phase tries to *find* a *correct* interpretation of a source sentence, and the generation phase need to *choose* a *good* expression among possible translations.

To choose a good expression, the following features are important for a language for the generation phase among the features discussed in section 2.3.1:

[4]From a linguistic point of view, we may say that there is not much difference between Japanese and English. At a practical level, however, it is better to use an *intuitive* structure suitable for each language, instead of a *sophisticated* internal structure applicable to *any* language, and difference between them should be handled in this transfer phase.

- *Using linguistic and non-linguistic heuristic information*: Linguistic rules only show *correct* structures. To generate a *good* translation, we need to use various heuristic information.

- *Choosing the most preferred output*: A mechanism to select the most preferred structure is more crucial in the generation phase than in the analysis phase.

- *Using word specific information*: Most of heuristic information is related to a word. The function to express word specific information is also important.

From this point of view, ACFG-like frameworks are also not suitable for the generation phase of a practical system, and we need to use a flexible grammar writing language.

2.3.4 Language for All Phases

So far, we have discussed grammar writing languages for each phase of the MT system. There are however two different approaches to design a grammar writing language:[5]

1. Using a *single* grammar writing language in all phases of machine translation (a general purpose grammar writing language)

2. Using *special* grammar writing languages for each phase

The grammar writing languages based on tree-to-tree transformation rules, such as ROBRA, are designed based on the idea 1. ATN and ACFG for an analysis phase are based on the idea 2.[6]

Since both ideas have advantages and disadvantages, EUROTRA, European machine translation project, has proposed that:

[5]In this section, we only consider frameworks for syntactic and semantic processing. Morphological processing (analysis and generation) is usually expressed with a different mechanism, because it needs a function to describe character-based processing.

[6]ATN can be used for the generation phase with its control structure [Simmons72]. DCG is also usable for the generation phase, because its implementation programming language Prolog has a feature of a reverse execution. However, we need an additional mechanism to use it for the generation [Dymetman88].

> A grammar writing software for the machine translation should provide linguists (grammar writers) of a machine translation system with a *generator* which generates highly specialized problem oriented systems [JohnsonR84].

This idea is similar to the one used in YACC (compiler generator [JohnsonS75]), which is used in the development of a programming language. It has the advantages:

- Many researchers who use different linguistic theories can cooperatively develop a single machine translation system.
- They can make an optimal grammar system for each phase in translation.

The idea of using *special* grammar writing languages and the idea of EUROTRA are useful for the MT system, because they allow grammar writers to use *optimal* description languages for each phase. However, they have the disadvantages:

- Grammar writers need to *learn* many grammar writing languages.
- Grammar writers tend to write their grammars with different *writing styles*, which cause a lack of understanding between grammar writers.
- Software developers have to develop many *similar* software tools to maintain grammars. Grammar writers also need to *learn* how to use them.

On the other hand, the general purpose grammar writing languages do not have these disadvantages. It is more desirable for the development of a practical MT system.

2.4 Software Tools for Grammar Development

We cannot finish the development of grammars (analysis, transfer and generation grammars) for the MT system in a short time, and we need to modify grammars continuously to improve the quality of translations. In addition, when we want to translate texts of a new field or a new style, we need to modify a large portion of grammars.

The development and improvement of grammars usually consists of the following four phases:

1. *Analysis of source texts*, to find linguistic and non-linguistic features of target sentences. We also need to find 'typographical' rules specific to the documents, such as the way of expressing mathematical formulas.

2. *Development and improvement of grammars*. Using features found in the target sentences, grammar writers add and modify grammar rules to improve the quality of translations.

3. *Translation test*. Grammar writers check their rules by translating sample sentences. During this translation test phase, various problems (including simple 'bugs' of rules) should be removed.

4. *Evaluation of translation results*. Even if grammar writers carefully verify their rules with sample sentences, grammars are usually still not perfect. They need to apply their grammars to a lot of sentences in target documents, and to evaluate the results of translation. When they find various incorrect translations, again they need to analyze source texts.

Since we cannot avoid repeating these four phases continuously, the efficiency to perform these phases is important for the development of an MT system. Thus we should develop various software tools to support this improvement process. In the rest of this section, we discuss functions necessary for the grammar development.

Analysis of Source Texts

When we run the translation system, we often find *unexpected* expressions in source texts. We should analyze the source texts from various points of view during the development and improvement of grammars. To analyze source texts, we need a large text database, and a software to handle it. Although the way to collect a large amount of texts is a research problem, tools to handle them are also important. We can use KWIC (KeyWord In Context) generator [Nagao83a], a retrieving program for the source sentences, and so on.

In addition to these, referring to translations by human translators is useful to compare source sentences and their translations. We need software tools to support this

comparison, including referring to other linguistic data (e.g. dictionaries) at the same time [Saito82].

Evaluation of Translation Results

When grammars become large and complex, we cannot easily find both good and bad effects of modifying a grammar. Even a simple modification to one part of a grammar sometimes makes unexpected effects to an other part. We can verify the improvement of the grammar by translating sample sentences whose results should be improved. However, to verify that the modifications do not have unexpected bad effects, we need to translate a lot of sentences and check the results of them.

Since checking all results is not easy by hand, we need flexible software tools for retrieving source sentences, internal structures (e.g. results of the analysis), and results of translation. A software tool, for instance, to find the difference of translation results among several versions of grammars is also useful [Yamano85].

'Debugger' for Grammars

To improve grammars according to the result of various analyses, we need a 'grammar debugger', similar to a conventional programming environment. The grammar debugger should have the following functions:

- *Tracer* to trace the behaviour of a translation process in detail
- *Breaker* to holt the execution at a specified point
- *Structure editor* to check and modify an internal structure such as a tree structure.
- *Window based system* to display various information, such as grammar rules, dictionary items, and so on.

2.5 Software for Dictionary Maintenance

When we develop large scale dictionaries of a practical MT system, there are various problems in their preparation, improvement and utilization, of which we are not aware

during the development of a small system. For effective utilization, for instance, we must design a dictionary maintenance system which has functions to manipulate various levels of dictionaries: a basic dictionary, a field specific dictionary, and a user specific dictionary.

In the rest of this section, we consider functions necessary for the development and maintenance of large dictionaries. Note that speed of the access method to look up large dictionaries is also important, because we cannot load all of them into a computer main memory. We thus need a special technique, such as loading words which are frequently used into a main memory [Slype79] [Hitaka82].

Extraction of Unknown Words

To improve dictionaries, we need a tool for the extraction of unknown words, the words which are not registered in the dictionaries. We may use, for instance, KWIC and outputs of morphological analysis. However, there are several problems in the recognition of a compound word in English, and the recognition of a word itself in Japanese. We need a good tool for the extraction of unknown words.

Registration Method

A method for the registration of words into dictionaries is important, because items in dictionaries of the MT system are complicated. There are two ways:

- Lexicographers fill special registration forms first. Then, punchers input them into the system (off-line method) [Nakamura84a] [Nakamura86d].

- Lexicographers directly input word data into the system with a guidance of a registration system (on-line method) [Kogure84] [Muraki84a].

The off-line method is useful for registering large dictionary data. The on-line method is suitable for improving the dictionary step-by-step. We thus need to support both methods for the effective improvement of dictionaries.

Verification of Dictionary Data

Tools to 'verify' dictionary data are necessary for the maintenance of dictionaries. In a practical level, we often catch a lot of *simple* registration errors: misspelling of symbols (e.g. category symbols and semantic markers), and omission of an item, for example. These errors make unnecessary works for lexicographers and grammar writers. We need, at least, a software tool to detect such *simple* format errors of the dictionary data [Katagiri85].

In addition, we need a tool to inspect relationships among words in dictionaries. The relationships among dictionary items are complicated in a *practical* system. These items often contain not only translation words, but also their derivations, words which are typically co-occurred with the entry words, for example. We need a program to verify consistency of dictionaries: whether the dictionary contains words referred to or not.

Other Utility Programs

We also need various utility programs to deal with a large dictionary: character code conversion programs, data format conversion programs [Nakamura84a] [Nakamura86d], programs for merging and splitting [Yamano85], and flexible retrieving programs.

2.6 Summary of this Chapter

In this chapter, we discussed the features of software environment neccessary for the development of a practical MT system. The software environment must have the following sub-systems:

- A grammar writing language which have the following features:
  - A framework to describe complex linguistic phenomena and to use heuristic information for translation, such as a tree-to-tree transducer.
  - A framework to express modularity of grammars and to select the most preferred output.

  - A language which allows to express word specific information in dictionaries.
  - A single language which is applicable to express all phases of the MT system (analysis, transfer, and generation)

- Various software for developing complex grammars.
- Various software for maintaining and utilizing large dictionaries.

In the following chapters, the author shows the grammar writing language and the software system to realize these requirements.

26 項欠

Chapter 3

Grammar Writing Language: GRADE

3.1 Language for Grammar Writing

A powerful grammar writing system was developed. This grammar writing system is called GRADE (GRAmmar DEscriber). GRADE allows the writing of grammars including analysis, transfer, and generation with the same expression. GRADE has powerful grammar writing facility. GRADE allows a grammar writer to control the process of a machine translation. GRADE also has a function to use grammar rules written in a word dictionary.

In this chapter, we first enumerate objectives necessary for the grammar writing framework of a machine translation system. Then, the characteristics of GRADE, which is designed to fulfil the objectives, are explained. Grammars written in GRADE and a software environment for GRADE language (GRADE system) are discussed in chapters 4 and 6.

The development of GRADE system started in 1982. GRADE system was used as the software of the machine translation project from Japanese into English and from English into Japanese, which is supported by the Japanese Government and called the Mu project [Nagao83b] [Nagao84] [Nagao85a] [Tsujii84] as explained in chapter 1.

3.2 Objectives

When we develop a machine translation system, the intention of a grammar writer should be accurately stated in the form of grammar rules. Otherwise, a good grammar system cannot be achieved. A programming language to write a grammar, which is composed of a grammar writing language, and a software system to execute it, is necessary for the development of a machine translation system [Boitet82].

If a grammar writing language for a machine translation system is to have a powerful writing facility, it must fulfill the following objectives:

Objective-1: A grammar writing language must be able to manipulate *linguistic characteristics* in Japanese and other languages. The linguistic structure of Japanese is largely different from that of English, for instance. Japanese does not restrict the word order strongly, and allows the omission of some syntactic components. When a machine translation system translates sentences between Japanese and English, a grammar writer must be able to express such characteristics.

Objective-2: It is desirable that a grammar writing language has the *same framework* to write grammars in the analysis, transfer, and generation phases. It should not be necessary for the grammar writer to learn several different formalisms for different stages of a machine translation.

Objective-3: There are many word specific linguistic phenomena in a natural language. A grammar writer must be able to add *word specific rules* to a machine translation system one after another to deal with word specific linguistic phenomena, and improve his machine translation system over a long period. The improvement of this kind is mainly done in the word dictionaries. Therefore, a grammar writing language must be able to handle grammatical rules written in word dictionaries.

Objective-4: There is a *natural sequence* in a translation process. For example, the parsing of simple noun phrases is executed in advance of a parsing of more complex noun phrases which contain sentential forms. A provisional parsing of compound sentences is executed before the parsing of complex sentences. When an appli-

cation sequence of grammatical rules are specified explicitly, a grammar writing system can execute the rules efficiently, because the system just needs to test the applicability of a restricted number of grammatical rules. In this way, a grammar writing language must be able to express different phases of a translation process in the expression explicitly.

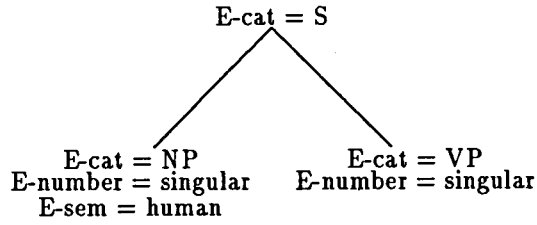
Objective-5: A grammar writing language must be able to resolve *syntactic and semantic ambiguities* in natural languages. But it must have some mechanisms to avoid a *combinatorial explosion.*

As discussed in chapter 2, many grammar writing languages have been developed for machine translation systems [Nakamura85c]:

- Augmented Transition Network (ATN) [Woods70], which is an augmentation of the push-down automaton.
- LINGOL [Pratt73] and DCG [Pereira80], based on Augmented Context Free Grammar (ACFG).
- SYSTEM-Q [Colmerauer70], PLATON [Nagao76] and ROBRA [Boitet79], based on tree-to-tree transformation.

Since push-down automaton and CFG formalism obviously cannot support our objectives 1-4, it is not easy to write a large grammar for a machine translation system using ATN and ACFG. Tree-to-tree transformation formalisms, such as PLATON, basically can support our objectives (especially objective 2). But SYSTEM-Q does not have a function to express a sequence in a translation process (objective 4). ROBRA does not allow writing grammar rules in word dictionaries (objective 3), and it does not have a mechanism to explicitly avoid a combinatorial explosion (objective 5). Then, we need a new formalism to support all these objectives for a *practical* machine translation system.

Keeping these points in mind, a new software system GRADE (GRAmmar DEscriber) for machine translation was developed, which is composed of the language specification for grammar writing and its executing system. GRADE is designed to support all these objectives as explained in the following sections.



E-cat : English category symbol
E-number : English Number (singular or plural)
E-sem : English semantic marker

Figure 3.1: Example of Annotated Tree in GRADE

3.3 Expression of Sentences for Processing

The form of data to express the structure of a sentence, during analysis, transfer, and generation process, has a strong effect on the framework of a grammar writing language. GRADE uses an *annotated* tree structure to represent a sentential structure during translation process. Grammar rules in GRADE are described in the form of *tree-to-tree transformation* with the annotation to each node.

The annotated tree in GRADE is a tree structure whose nodes have lists of property names and their values. Figure 3.1 shows an example of the annotated tree. The left daughter node of this tree contains the information:

- English category is NP (noun phrase).
- Its number is singular.
- Its semantic marker is human.

The annotated tree can express various information such as syntactic category, number, semantic marker, and other things, and there is no limitation in the number of properties.

Note that the annotated tree may contain a *flag* in its node, which controls the process of a translation, similar to a flag in a conventional programming language. For

example, in a grammar of generation, there is a grammar rule which is applied to all nodes in the annotated tree whose processing is not finished. In such a case, a grammar rule checks the DONE flag in the annotation whether it is processed or not, and sets T to the newly processed ones.

3.4 Rewriting Rule in GRADE

The basic component of a grammar writing language is a *rewriting rule.* The rewriting rule in GRADE transforms one annotated tree into another annotated tree. Because the tree-to-tree transformation by this rewriting rule is very powerful, it can be used in the grammars of analysis, transfer and generation phases of a machine translation, as stated in the objective 2.

A rewriting rule in GRADE consists of 2 parts:

1. A declaration part, in which information about a rewriting rule are defined.
2. A transformation part, in which *conditions* (so-called left hand side of a rewriting rule) and a *result* of the rule application (so-called right hand side of a rewriting rule) are written.

An example of the rewriting rule, CHECK_NOUNS, is is shown in figure 3.2.[1]

3.4.1 Declaration Part

The *declaration part* has the following four components:

1. Directory entry part.
2. Property definition part.
3. Variable declaration part.
4. Matching instruction part.

[1]In figures of this chapter, lower case letters is used for expressing GRADE keywords, such as 'if.' The GRADE translator, which converts GRADE rules into internal forms for execution discussed in chapter 6, does not distinguish lower case letters and upper case letters.

```
CHECK_NOUNS.rr;
                                                    /* Declaration Part */
  directory_entry;
    owner(J.NAKAMURA); version(V01L02); last_update(84/4/19);
  prop_def;
    J_CAT: type(u), value(NOUN VERB ADJ);
  var_import;
    @NONSNP;
  var_init;
    @UME;
  matching_instruction;
    level(3,10); left_to_right; bottom_to_top; depth;
    order(2,noskip); tree;
                                                  /* Transformation Part */
  matching_condition;
    %((X0  X1 BKK1 X2 BKK2 X3 BKK3 X4 #2));
      X0.NONSNP = @NONSNP.VAL;
      X1= 'N' | 'NP'; X2= 'N' | 'NP';
      X3= 'N' | 'NP'; X4= 'N' | 'NP';
  substructure_operation;
    if X1.J_PASS='YES';
      then @A<=call_sg(CHECK_PATTERN
                %((X2 BKK2 X3 BKK3 X4 #2) list);
      else @A<=call_sg(FOUR_OR_MORE_NOUNS
                %((X1 BKK1 X2 BKK2 X3 BKK3 X4 #2) list);
      end_if;
    @A.UME <= @UME.VAL;
  creation;
    if X1.J_PASS='YES';
      then %((X0 X1 BKK1 @A));
      else %((X0 @A));
      end_if;
  end_rr.CHECK_NOUNS;
```

Figure 3.2: Example of Rewriting Rule

Directory Entry Part

A *directory entry part* of a rule contains the name of a grammar writer who writes the rewriting rule, a version number of the rewriting rule, and the last date of the revision. In figure 3.2, the directory entry part shows that

> the rule CHECK_NOUNS is written by J. NAKAMURA on 84/4/19 and the version of this rule is V01L02.

This part is not used at the execution time of the rewriting rule. A grammar writer, however, is able to see the information written in this part by using the help facility of the GRADE system, for example, when he finds something wrong in the rewriting rule [Nakamura85a]. This facility is necessary for a machine translation system, when many grammar writers cooperate to develop a grammar.[2]

Property Definition Part

In a *property definition part*, a grammar writer declares property names and their values which he is using. In figure 3.2,

> J_CAT (Japanese Category Symbol) is used as a property name and its value is one of NOUN, VERB, and ADJ.

A grammar writer can remember what properties he uses. This part is useful to check the consistency of rewriting rules, when the grammar becomes very large.

Variable Declaration Part

In a *variable declaration part*, a grammar writer declares the names of global variables (var_import) and local variables (var_init). In figure 3.2, this part shows that

- in var_import, @NONSNP is a global variable, which should be declared in other rule that calls this rule.
- in var_init, @UME is a local variable, which is used in this rule or rules called by this rule.

[2]Details of the software tools to maintain grammar rules are explained in chapter 6.

A grammar writer can use variables to control rule applications like registers in Augmented Transition Network [Woods70].[3]

Matching Instruction Part

In a *matching instruction part*, a grammar writer specifies the mode of application of the rewriting rule to an annotated tree, which is explained in section 3.4.3.

3.4.2 Transformation Part

The *transformation part* specifies the tree-to-tree transformation in the rewriting rule, and has the following three parts.

1. Matching condition part: where the condition of a structure and the property values of an annotated tree is described.
2. Substructure operation part: which specifies operations for the annotated tree that has matched with the condition written in the matching condition part.
3. Creation part: which specifies the structure and the property values of the transformed annotated tree.

Matching Condition Part

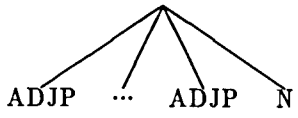
The *matching condition part* specifies the condition of the structure and the property values of the annotated tree. The matching condition part allows a grammar writer to specify not only a rigid structure for the annotated tree, but also

- structures which may repeat several times (`any, length`),
- structures which may be omitted (`optional`),
- structures that the order of sub-structures is free (`disorder`).

This function is used to manipulate the linguistic characteristics, especially in Japanese, which are discussed in the objective 1.

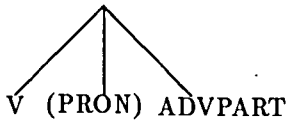
For example, a grammar writer can express the following structures:

[3]The example of the usage of variables is shown in section 3.8.2.




```
matching_condition;
  %(ADJPS N);
    ADJPS: any(%(ADJP));
```

Figure 3.3: Expression of Tree Structure with 'any'



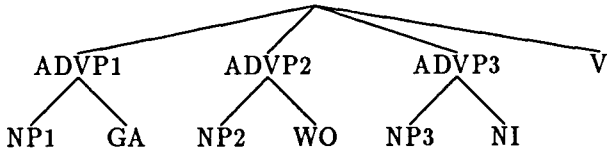

```
matching_condition;
  %(V PRON ADVPART);
    PRON: optional;
```

Figure 3.4: Expression of Tree Structure with 'optional'

- The structure in Japanese, in which adjective phrases (ADJP) repeat arbitrary number of times and then a noun (N) follows them is written as the expression 'any' in figure 3.3.

  Note that 'any(%(ADJP))' is an abbrevation of 'length(0,100,%(ADJP))', in which a grammar writer may specify the number of repetition. GRADE also allows a grammar writer to abbrevate 'any' which does not have constraints (e.g. the restriction for a part-of-speech) to '#.' If he needs to express the pattern: *so ... that*, he may write '%(SO #1 THAT)' in his rule.

- The structure in English, like a combination of a verb (V) and an adverbial particle (ADVPART) in this sequence with or without a pronoun (PRON) in between is expressed as in figure 3.4 (optional). GRADE matches this pattern to '*put it on*' and '*put on*', for instance.




```
matching_condition;
  %(A1 A2 A3 V);
    A1, A3: disorder;
    A1: %((ADVP1 NP1 GA));
    A2: %((ADVP2 NP2 WO));
    A3: %((ADVP3 NP3 NI));
```

Figure 3.5: Expression of Tree Structure with 'disorder'

```
matching_condition;
  %(NP  VP);
  NP.NUMBER = VP.NUMBER;
```

Figure 3.6: Condition of Number Agreement between Subject Noun and Verb

- A typical Japanese sentential structure in which three adverbial phrases (ADVP), each composed of a noun phrase (NP) and a case particle (GA, WO, or NI) precede a verb (V) as in no particular order, is written as the expression (disorder) in figure 3.5. Note that A1, A2 and A3 are used as labels to express the tree structure. These are not regared as 'non-terminal' symbols.

The matching condition part allows a grammar writer to specify conditions about property names and property values for the nodes of the annotated tree. A grammar writer can compare not only a property value of a node with a constant value, but also values *between two nodes* in a tree. For example, the number agreement between a subject noun and a verb is written as the expression in figure 3.6. In this case, NP.NUMBER and VP.NUMBER specify the values of the property NUMBER of the node NP and VP respectively.

```
substructure_operation
  if NP.NUMBER = 'SINGULAR';
     then DET.LEX <= 'A';
     else DET.LEX <= 'NIL';
     end_if;
```

Figure 3.7: Example of Sub-structure Operation Part

```
creation;
  %((S NP VP));
```

Figure 3.8: Example of Creation Part

Substructure Operation Part

The *substructure operation part* specifies operations on the annotated tree which has matched with the matching condition part. The substructure operation part allows a grammar writer to set a property value to a node, and to assign a tree or a property value to a variable declared in the variable declaration part. It also allows him to call a subgrammar, a subgrammar network, a dictionary rule, a built-in function, or a LISP function, which will be explained in section 3.5, 3.6, and 3.7.

In addition, a grammar writer can write a conditional operation by using the *if-then-else* form, to control the translation process. For example, an operation to set 'A' to the lexical unit of the determiner node (DET.LEX), if the number of the NP node is SINGULAR, is expressed as in figure 3.7.

Creation Part

The structure and the property values of the transformed annotated tree is written in the *creation part*. The transformed tree is described by node names such as NP and VP which are used in the matching condition part or the substructure operation part. A creation part to create the tree whose top node is S and which has an NP subtree and a VP subtree is written as shown in figure 3.8.

```
ABCD.rr;
  matching_instruction;
    level(0,100); left_to_right; bottom_to_top; depth;
    order(2,noskip); tree;
  matching_condition;
    %((A #1 B C D #2));
  creation;
    %((E #1 (A B C D) #2));
  end_rr.ABCD;
```

Figure 3.9: Example of Rewriting Rule Using Matching Instruction Part

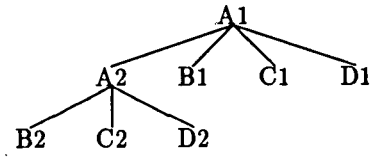


Figure 3.10: Input Tree Structure for the Rewriting Rule

3.4.3 Matching Instruction Part

There is a case that a grammar writer needs to apply a transformation to *all* subtrees in a sentential structure. For example, when a grammar writer wants to decide determiners of all noun phrases in a sentence, he needs not only a rule to choose a determiner of one noun phrase, but also an algorithm to traverse all subtrees in a sentential structure. The rule for one noun phrase is written in the transformation part of a rewriting rule, and the order of traversal in a tree is specified in a *matching instruction part.*

Let us consider that a rewriting rule shown in figure 3.9 is applied to a tree in figure 3.10. The transformation part of this rule makes a new node `E` over a tree whose top node is `A` and whose subtrees are `B`, `C`, and `D`.

GRADE applies this transformation to *all* subtrees in the tree, because the level of traversing is indicated by '`level(0,100)`' in the matching instruction part of the rewriting rule.[4] The input tree is traversed from left to right and from bottom to top in

[4]'`level(0,100)`' means that GRADE applies the transformation part from the root node (0) to the 100th daughter nodes (if they are exist).

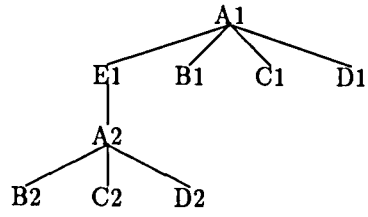


Figure 3.11: Result of Single Application of the Rewriting Rule

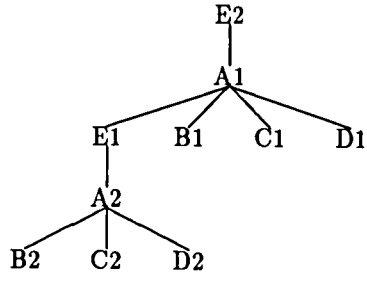


Figure 3.12: Final Result of Application of the Rewriting Rule

a depth-first order. This order of traversal is specified by the keywords: `left_to_right`, `bottom_to_top`, and `depth`.

Therefore the application of the rewriting rule is the following:

- The tree in figure 3.10 is first transformed into a tree in figure 3.11, because the subtree whose top node is `A2` is an input tree to the transformation part and is transformed into a tree whose top node is `E1`.
- Then the tree in figure 3.11 is changed to the tree in figure 3.12, because, at this time, the subtree whose top node is `A1` is transformed.

There are eight types of the traverse paths, which are the combinations of

1. left-to-right or right-to-left.
2. bottom-to-top or top-to-bottom.

3. depth-first or breadth-first.

A grammar writer is able to choose one of the eight types to control the order of his rule application.

3.5 Control of Grammar Rule Applications

A grammar writing language must be able to describe the detailed phases of a translation process in the expression explicitly (objective 4). GRADE allows a grammar writer to divide a whole grammar into several parts. Each part of the grammar is called a *subgrammar*. A subgrammar may correspond to a grammatical unit such as the parsing of a simple noun phrase and the parsing of a compound sentence. A whole grammar is then described by a network of subgrammars. This network is called a *subgrammar network*. A subgrammar network allows a grammar writer to control the process of a translation in detail.

When a subgrammar network in the analysis phase consists of a subgrammar for a noun-phrase (`SG1`) and a subgrammar for a verb-phrase (`SG2`) in this sequence, the executor of GRADE first applies `SG1` to an input sentence, then applies `SG2` to the result of an application of `SG1`.

This control structure makes it possible to use 'a procedural approach' [Tsujii84] for resolving the ambiguities (objective 5) and selecting the most preferable reading of a sentence.

3.5.1 Subgrammar

A subgrammar consists of a set of rewriting rules. It is important to decide the order of rule application in a set of rewriting rules. There are two ways as follows:

- One way is that the system puts the same priority on *all* rewriting rules, and makes all possible tree structures which satisfy the constraints of each rewriting rule independently, like context free rules. This is a suitable option for a linguistic research, but not useful for a machine translation system, because it usually makes so many translations that the system and users, especially post-editors cannot manage them.

- The other way is that the system puts a *priority ordering* in rule applications, makes only one possible tree structure at one time, and generates other possible tree structures by using backtracking method, if necessary. This option allows a grammar writer to use a *heuristic* knowledge concerning the preference of rewriting rules. When a grammar writer develops a machine translation system, he must use much heuristic knowledge to output a translation which expresses the first reading of the source sentence.

Therefore, rewriting rules in a subgrammar of GRADE has a *priority ordering* in their application. In other words, the n-th rewriting rule in a subgrammar is tried before the $(n+1)$-th rule.

A grammar writer can specify four types, `order(1)`–`order(4)`, of application sequence of rewriting rules in the subgrammar. Let us assume the situation:

- A set of rewriting rules in the subgrammar is composed of RR_1, RR_2, ..., and RR_n.
- RR_1, ..., and RR_{i-1} failed in the application to an input tree.
- RR_i succeeded in it as the first application.

Then each of four types works as follows:

- When a grammar writer specifies the first type, which is called `order(1)`, the effect of the subgrammar execution is the application of *RRi* to the input tree, and the control quits from this subgrammar without applying other rewriting rules (RR_{i+1}, ..., RR_n).
- When a grammar writer specifies the second type, which is called `order(2)`, the executor of GRADE tries to apply RR_{i+1}, ..., RR_n to the result of the application of RR_i. So, `order(2)` means that rewriting rules in the subgrammar are sequentially applied to an input tree.
- The third and fourth type, which are called `order(3)` and `order(4)`, are the iteration type of `order(1)` and `order(2)` respectively. So, the executor of GRADE

tries to apply rewriting rules until no rewriting rule is applicable to the annotated tree.[5]

A subgrammar consists of *matching instruction part*, which specifies the application type order(1)--order(4), and *rr_in_sg part* (rewriting rules in subgrammar), which is a ordered list of rewriting rule names. Figure 3.13 shows an example of a subgrammar.

When this subgrammar is applied to an annotated tree, the executor of GRADE first tries to apply the rewriting rule CANDIDATE_OF_NOUNS_1 to the input tree. If the application of this rule succeeds, the input tree is transformed to the result of the application of the rewriting rule CANDIDATE_OF_NOUNS_1. Otherwise, the input tree is not modified. In either case, the executor of GRADE next tries to apply the rewriting rule UP_NP_TO_PNP to the input tree. The executor continues such a process until the application of the last rewriting rule CANDIDATE_OF_NOUNS_2 is finished.

3.5.2 Subgrammar Network

A subgrammar network describes the application sequence of subgrammars. The specification of a subgrammar network consists of the following five parts:

1. Directory entry part.
2. Property definition part.
3. Variable declaration part.
4. Entry part.
5. Network part (Node specification and Link specification).

Directory Entry Part

A *directory entry part*, and a *property definition part* are defined in the same way as in a rewriting rule. They also used as the default declaration in rewriting rules which are called by the subgrammar network.

[5]These two types (order(3) and order(4)) may cause *infinite loop* of the execution. GRADE leaves a grammar writer to stop the execution, although the GRADE executor, which is explained in section 6.2, has a function to detect the possibility of *infinite loop*. See section 3.8.1.

```
/* Definition of the subgrammar */
SEARCH_CANDIDATE_OF_NOUNS.sg;
  matching_instruction;
    order(3);
  rr_in_sg;
    CANDIDATE_OF_NOUNS_1;
    UP_NP_TO_PNP;
    CANDIDATE_OF_NOUNS_2;
  end_sg.SEARCH_CANDIDATE_OF_NOUNS;

/* Rewriting rules called by the subgrammar */
CANDIDATE_OF_NOUNS_1.rr;
  matching_condition;
    %((NP #));
      NP.J_DEEP_CASE='PARAELEMENT';
  substructure_operation;
    @A<=call_sgn(CUT_NOUNS %(#) list);
  creation;
    %(@A);
  end_rr.CANDIDATE_OF_NOUNS_1;

UP_NP_TO_PNP.rr;
  matching_instruction;
    order(2,skip);
  matching_condition;
    %((X # Y));
      X.J_CAT='NP';
      X.J_PARA='T';
      Y.J_CAT='N'|'NP';
  creation;
    %((X # Y));
      X.J_LEX<=Y.J_LEX;
      X.J_SEM<=Y.J_SEM;
      X.J_N<=Y.J_N;
  end_rr.UP_NP_TO_PNP;
```

Figure 3.13: Example of Subgrammar and Rewriting Rules

```
PRE.sgn;
  directory_entry;
    owner(J.NAKAMURA); version(V02L05); last_update(83/12/25);
  var_init;
    @PRE_FLAG init(T);
  entry;
    START;
  network;
    START: PRE_STEP_1.sg;
    LOOP : PRE_STEP_2.sg;
    A:     PRE_STEP_3.sg;
    B:     PRE_END_CHECK.sg;
             if @PRE_FLAG; then goto LOOP; else goto LAST;
    LAST:  PRE_STEP_4.sg;
             exit;
  end_sgn.PRE;
```

Figure 3.14: Example of Subgrammar Network

Variable Declaration Part

A *variable declaration part* is also the same as in a rewriting rule. Variables are used to control the transition of the subgrammar network. Variables are also referred to in a *link specification part*, which will be described later. An *entry part* indicates a start node of the network. A *network part* is a body of the subgrammar network.

Network Part

The *network part* specifies the network structure of subgrammars, and consists of node specifications and link specifications. The *node specification* has a label and a subgrammar, or a subgrammar network name, which is called when the node gets the control of the processing. The *link specification* defines the transition between nodes in a subgrammar network. The link specification checks the value of a variable which is set in a rewriting rule, and decides which node label will be processed next.

Figure 3.14 shows an example of a subgrammar network. When the executor of GRADE applies this subgrammar network to an input tree, the executor checks the variable declaration part, then puts a new variable @PRE_FLAG in a stack, and sets T to @PRE_FLAG as an initial value. After that, the executor checks the entry part and finds

```
CASE_FRAME.rr;
  var_init; @S;
  matching_condition;
    %(NP1 V NP2 PP);
  substructure_operation;
    @S <= call_dic(V.LEX ANALYSIS %(NP1 V NP2 PP));
                /* V.LEX : key for dictionary look-up        */
                /* ANALYSIS: name of the dictionary rule set */
  creation;
    %(@S);
  end_rr.CASE_FRAME;
```

Figure 3.15: Example of Rewriting Rule Calls Dictionary Rule

the label of the start node START in the network.

3.6 Grammar Rules in Word Dictionaries

As discussed in objective 3, a grammar writer often needs to write a specific rule for each word. GRADE allows a grammar writer to write word specific grammar rules as a subgrammar in an entry of word dictionaries of a machine translation system. A subgrammar written in a dictionary entry is called a *dictionary rule.* The dictionary rule is specific to a particular word in the dictionary.[6]

When *call_dic* operation in the substructure operation part is executed, the dictionary rule is retrieved with an entry word and a rule identifier as keys, and is applied to the annotated tree which is specified by a grammar writer. Figure 3.15 shows an example of a rewriting rule which calls a dictionary rule. In this case, a dictionary rule which is written in an entry of a verb as indicated by V.LEX (the value of the lexical unit of the verb), and whose name is ANALYSIS, is applied to the sequence of NP1, V, NP2, and PP (noun phrase 1, verb phrase, noun phrase 2, and prepositional phrase). Then the result of the application of the dictionary rule is assigned to the variable @S.

Note that this *call_dic* function may be used as *conditional branching*, instead of calling a dictionary rule. The implementation of *call_dic* just makes a rule name from the arguments and calls it. Therefore, a grammar writer may use *call_dic* to dispatch

[6]Examples of the dictionary rule are shown in section 5.5.2.

the control of processing with any property value, such as the values of deep-case.

3.7 Resolving Ambiguities

A grammar must be able to resolve the syntactic and semantic *ambiguities* in natural languages (objective 5). GRADE allows a grammar writer to collect all the results of possible tree-to-tree transformations by a subgrammar (not a whole grammar), when it encounters the ambiguities. This function can be used to localize non-deterministic processing for handling ambiguities, and to avoid a *combinatorial explosion.*

For instance, let us assume that a grammar writer writes a subgrammar which contains two rewriting rules to analyze the case frame of a verb:

- A rewriting rule is the rule to construct `VP` (verb phrase) from `V` and `NP` (a verb and a noun phrase):

  `VP` ⟶ `V NP`

- A rewriting rule to construct `VP` (verb phrase) from `V`, `NP` and `PP` (a verb, a noun phrase, and a prepositional phrase):

  `VP` ⟶ `V NP PP`

When he specifies *nondeterministic-parallel mode* for the subgrammar, the executor of GRADE applies both rewriting rules to an input tree, constructs two transformed trees, and merges them into a new tree whose top node has a special property `PARA`. The top node of this structure is called a *para special node*, its subtrees are the transformed trees by the rewriting rules. Figure 3.16 shows an example of this mode and a para node.

A grammar writer can select the most appropriate one from the subtrees under a para special node. A grammar writer is able to use built-in functions in the substructure operation part to choose the most appropriate one:

- `map-sg`, `map-sgn`: To apply a subgrammar (network) to all sub-trees.
- `sort`: To sort the order of sub-trees by values of a specified property name.

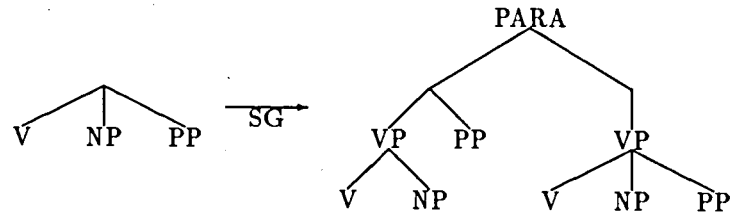


Figure 3.16: Example of PARA Node

```
substructure_operation;
  @X <= call_dic(V.LEX CASE-FRAME %(N NP PP));
  @X <= call_built(map-sg %(@X) tree EVALUATE_CASE_FRAME);
  @X <= call_built(sort %(@X) tree SCORE);
  @X <= call_built(cut %(@X) tree 1);
  @X <= call_built(injection %(@X) tree 1);
```

Figure 3.17: Example of Built-in Functions

- cut: To select n-th sub-tree and to remove others.
- injection: To remove a para special node.

Figure 3.17 shows an example to use these built-in functions. The execution of this substructure operation part is as follows:

- First the executor of GRADE applies to the tree the dictionary rule written in a word entry which is the value of V.LEX (lexical unit of verb), and sets the result to the variable @X. When the *nondeterministic-parallel* mode is used in the dictionary rule, which is expressed as a subgrammar, the value of @X is the tree whose root node is a para special node.
- After that, the executor calls the built-in function map-sg to apply the subgrammar EVALUATE_CASE_FRAME to each subtree of the value of @X, and sets the result to @X again. The subgrammar EVALUATE_CASE_FRAME computes the evaluation score and sets the score to the value of the property SCORE in the root node of the subtrees.

- Next, the executor calls built-in function `sort`, `cut`, and `injection` to get the subtree whose score is the highest one among the subtrees under the para special node. This tree is then set to `@X` as the most appropriate result of the dictionary rule.

The para special node is treated as the same as the other nodes in the current implementation of GRADE. A grammar writer can use the para node as he wants, and can select a subtree under a para node during later grammar rule applications.

3.8 GRADE and Formal Language

3.8.1 Rewriting Rules

Let us consider the GRADE language from a view point of the formal language theory. Formally, a language L of grammar G can be expressed by a 4-tuple:

$G = (S, V_t, V_n, R)$ where

V_t: a set of terminal symbols,

V_n: a set of non-terminal symbols,

S: a start symbol, which is a member of V_n,

R: a set of rewriting rules.

Although the GRADE language is not formalized with this framework directly, the rest of this section discusses symbols, rewriting rules, and the parsing algorithm of the GRADE language.

Symbols

Terminal symbols V_t, and non-terminal symbols V_n are not distinguished in the GRADE language. These should be classified by grammar writers who use GRADE.

The start symbol S is also not defined in the GRADE language. This means that GRADE always *accepts* an input, although grammar writers may distinguish *accepted* and *not-accepted* states. This is useful for the MT system which needs to output, at least, fragmental translations, even if the system cannot analyze input sentences as a whole.

```
TYPE0_EG1.rr;
  matching_condition;
    %((? #1 D E #2));
  creation;
    %((? #1 A B C #2));
end_rr.TYPE0_EG1;
```

Figure 3.18: Rewriting Rule of Type 0

```
TYPE0_EG2.rr;
  matching_condition;
    %((? #1 B C D #2));
    C: optional;
  creation;
    %((? #1 A #2));
end_rr.TYPE0_EG2;
```

Figure 3.19: Rewriting Rule Equivalent to Two Type 0 Rules

Rewriting Rules

Rewriting rules in GRADE are equivalent to type 0 rewriting rules. For example, a rule of type 0:

A B C $\longrightarrow$ D E

is expressed with the GRADE grammatical rule shown in figure 3.18.[7]

When a grammar writer uses the function of *optional*, for instance, it correspond to merging several type 0 rewriting rules into a single GRADE rewriting rule. Two rewriting rules:

A $\longrightarrow$ B C D and A $\longrightarrow$ B D

are written in the single GRADE rewriting rule shown in figure 3.19.

Although rules in the GRADE language are equivalent to rewriting rules in type 0 language, there are differences:

[7]In the GRADE language, symbols written in the matching condition and the creation part may not be 'non-terminal' symbols. They are labels to identify a tree structure. However, the GRADE translator regards them as 'non-terminal' symbols by default. Hence the label 'D' in figure 3.18, for example, expresses a tree structure whose top node contains 'D' as the value of the user specified propery name, such as 'category.'

```
TYPE0_EG3.rr;
  matching_condition;
    %((? #1 A B C #2));
  creation;
    %((? #1 D E #2));
end_rr.TYPE0_EG3;
```

Figure 3.20: Rewriting Rule of Type 0 for Generation

- GRADE rewriting rules have directionality. We cannot use a rewriting rule for analysis and generation at the same time. If we want to use the rule

  A B C $\longrightarrow$ D E

  for generation, we should express it as shown in figure 3.20.

- GRADE rewriting rules express the tree structure of the result of their applications in the creation part. A grammar writer is allowed to specify the result by himself.

Parsing

A *good* parsing algorithm of the type 0 language is not known yet. Theoretical problems caused by this are:

- Does the parser recognize all possible results which are allowed with grammar rules?[8]

- Does the parser stop its execution for any input trees?

The answers of GRADE are negative. However, the first problem conflicts with the procedural grammar approach. The GRADE system is designed to output a single result for a sentence, and the grammars of the Mu project do not allow the generation of multiple results.

[8]The efficency of a parsing algorithm becomes also important, when we need to generate all possible results.

Although the second problem is not solved theoretically, the GRADE executor has a 'counter' for each rule. If the GRADE executor detects the possibility of an infinite number of rule applications, it halts execution.[9]

3.8.2 Subgrammar and Subgrammar Network

Subgrammar networks and subgrammars in the GRADE language can express the deterministic finite automata in a straightforward way. Let us consider a deterministic finite automaton:

$$M = (K, \Sigma, \delta, q_0, F),$$
$$\Sigma = \{0, 1\},\ K = \{q_0, q_1, q_2\},\ F = \{q_0\}$$
$$\delta(q_0, 0) = q_1, \delta(q_0, 1) = q_2,\ \delta(q_1, 0) = q_0, \delta(q_2, 1) = q_0$$

where K: a set of states, Σ: a set of input characters, δ: a transition function, q_0: an initial state, F: a set of final states.

This automaton is expressed with the GRADE rules in figure 3.21. The link specifications labeled with `Q0`, `Q1`, and `Q2` (lines 5–14) correspond to the states q_0, q_1, and q_2 respectively. The variable `@SYM` is used to express an input character 0 or 1, and is set in the rewriting rule `DFA_READ.rr` (lines 22–27). The `goto` statements (e.g. line 6) describe the transition function δ. The subgrammar `DFA_READ.sg` (lines 18–21) is a dummy to call the rewriting rule `DFA_READ.rr`.

Thus we can express a deterministic finite automaton with the subgrammar network in a straightforward way. Note that the rewriting rules to read an input string may refer to the whole string, instead of the first character. This allows the transition function to be expressed in a more flexible way.

3.9 Summary of this Chapter

In this chapter, we first discussed the objectives required in a grammar writing system for a practical machine translation system. The objectives are:

[9]The GRADE executor has an option to skip rule applications, instead of halting execution.

```
1   DFA.sgn;
2     var_init;                  @SYM init(nil);
3     entry;                     Q0;
4     network;
5       Q0: DFA_READ.sg;
6         if @SYM = '0';       then goto Q1;
7         else if @SYM = '1'; then goto Q2;
8         else                      goto ACCEPT;
9       Q1: DFA_READ.sg;
10        if @SYM = '0';       then goto Q0;
11        else                      goto NOT_ACCEPT;
12      Q2: DFA_READ.sg;
13        if @SYM = '1';       then goto Q0;
14        else                      goto NOT_ACCEPT;
15      ACCEPT: DFA_ACCEPT.sg;     exit;
16      NOT_ACCEPT: DFA_NOT_ACCEPT.sg; exit;
17  end_sgn.DFA;

18  DFA_READ.sg;
19    sg_mode;                   order(1);
20    rr_in_sg;                  DFA_READ;
21  end_sg.DFA_READ;

22  DFA_READ.rr;
23    var_import;                @SYM;
24    matching_condition;        %((? SYM #1));
25    substructure_operation;      @SYM <= SYM.LEX;
26    creation;                  %((? #1));
27  end_rr.DFA_READ;
```

Figure 3.21: Subgrammar Network to Express the Finite Automaton

1. A grammar writing language must be able to manipulate *linguistic characteristics* in Japanese and other languages.

2. A grammar writing language should have the *same framework* to write grammars in the analysis, transfer, and generation phases.

3. A grammar writing language must be able to handle grammatical rules written in word dictionaries.

4. A grammar writing language must be able to express different phases of a translation process in the expression explicitly.

5. A grammar writing language must be able to resolve *syntactic* and *semantic ambiguities* in natural languages.

Based on this discussion, the author developed the grammar writing system GRADE. GRADE has the following features to fulfil the objectives:

1. Rewriting rules are expressions in the form of *tree-to-tree transformation* with annotations for each node. (objective–1, 2)

2. Rewriting rules have *powerful* capability to handle sophisticated linguistic phenomena. (objective–1)

3. A grammar can be divided into several parts as subgrammars, and each part is linked together as *subgrammar networks*. (objective–4, 5)

4. A subgrammar (a set of rewriting rules) can be written in the dictionary entries to express word specific linguistic phenomena (*dictionary rule*). (objective–3)

5. *Para special nodes* are provided in trees for embedding ambiguities. (objective–4)

Although we discussed the power of GRADE from the view point of formal language theory in section 3.8.1, we need to clarify the effectiveness of GRADE in development of a practical machine translation system. In the next chapter, we discuss the grammar written in GRADE to demonstrate the effectiveness of GRADE.

54 項欠

Chapter 4

Grammars for Machine Translation

4.1 Problems of MT Grammars

In chapter 3, characteristics of the grammar writing system GRADE are described. To demonstrate that GRADE is useful to develop grammars of a *practical* machine translation system, we discuss grammars written in GRADE in this chapter.

For the development of a *practical* machine translation system, we cannot directly apply typical linguistic models proposed by (computational) linguists, such as GB, GPSG, and LFG [Sells85]. This difficulty is caused by the characteristics of sentences we want to translate with a *practical* system:

- Complexity of sentences: In many cases, complexity of the structure of sentences we want to translate is different from that of sentences which computational linguists use to discuss their models. For example, the sentences contains complex coordinate clauses of noun phrases and verb phrases, ellipses of constituents, and so on.

- Ambiguities of sentences: Sentences have a lot of ambiguities locally and globally. Global and intrinsic ambiguities, such as PP (Prepositional Phrase) attachment, are typical problems, of course. However, ambiguities of parts-of-speech, for example, are also crucial for a *practical* system.

- Complexity of a transfer phase: There are many word specific rules at the transfer phase. A high quality machine translation system needs flexibility to handle such word specific rules.

To overcome these difficulties in the development of a *practical* system, the grammars developed by the Mu project use the following functions of GRADE:

- Subgrammars and subgrammar networks to control the translation process: Using this mechanism, the Mu project has developed the grammars based on the idea of *procedural grammar* [Tsujii84].
- Powerful Pattern Matching function: To describe complex linguistic phenomena, the grammars of the Mu project use the flexible pattern matching function effectively.
- Para special node to localize ambiguities: For the disambiguation, the grammars use a lot of heuristic rules. Even if heuristic rules cannot solve the ambiguities, the grammars use the *para special node* to localize the ambiguities.
- Dictionary rules: Many word specific rules are used in the grammars. These are expressed as *dictionary rules* for clearness of the grammars and efficiency of the processing.

In this chapter, we discuss both theoretical aspect of these ideas and their practical applications in the Mu project.

4.2 Solutions for Problems of MT grammars

4.2.1 Problems Caused by Ambiguity of Parts-of-Speech

Combinatorial explosion, especially the one caused by the ambiguity of English parts-of-speech, is one of the most crucial problems in a *practical* system. In a *model* system, however, the ambiguity of parts-of-speech is not considered seriously, because it usually uses a small scale dictionary in which words have parts-of-speech necessary to the explanation of the model. In addition to it, many of them are automatically solved, when we use, for example, a *context free grammar.* However, the situation becomes more complex in a *practical* system.

Let us consider a sentence:

The authors consider the processes.

```
? <?> <ROOT>
|--ART <the>
|--N <author> <> <COM>
|--V <consider> <USAGE_PARA> <V>
|  |--V <consider> <> <V>
|  |--V <consider> <> <V>
|  |--V <consider> <> <V>
|--ART <the>
|--? <?> <PARA>
|  |--N <process> <USAGE_PARA> <COM>
|  |  |--N <process> <> <COM>
|  |  |--N <process> <> <COM>
|  |  |--N <process> <> <COM>
|  |  |--N <process> <> <COM>
|  |--V <process> <> <V>
|--SYMB <.> <> <PUNC>
```

```
|--N <process> <USAGE_PARA> <COM>
```

menas that:

1st field:	part-of-speech	N (noun)
2nd field:	lexical unit	process
3rd field:	special attribute	USAGE_PARA
4th field:	sub-category	COM (common noun)

Figure 4.1: Result of Morphological Analysis for Simple Sentence

The most simple evaluation of the ambiguity of this sentence is calculated by multiplying the number of parts-of-speech of each word. With this evaluation, this sentence has only 2 times ambiguity, because the word '**processes**' has a possibility to be a 'verb' (third person singular present form) or a 'noun' (plural form), and other words have single part-of-speech.

Figure 4.1 shows the result of a morphological analysis by the Mu project. In this figure, the first field of the tree node shows a part-of-speech ('?' means 'unknown' or 'undefined'). The second field is a lexical unit. The third field is a special attribute to mark a non-syntactic node. In this example, 'PARA' means that the node have alternatives of parts-of-speech, and 'USAGE_PARA' means that the node has several usages. These are also expressed as tree structures. The fourth field is a sub-category of the word. For example, 'author' is a Noun and its sub-category is a COMmon noun.

Table 4.1: Part-of-Speech and Usage Ambiguity of Sentences in Abstracts

Number of Sentences:	2,999	sentences
Average Sentence Length:	18.4	words
Ambiguity of Part-of-Speech par Word:	1.27	times
Ambiguity of Part-of-Speech par Sentence:	86.7	times
Ambiguity of POS and Usage Pattern par Word:	1.58	times
Ambiguity of POS and Usage Pattern par Sentence:	4,705	times

In this sentence, there is no problem about both the computational complexity and the correct disambiguation, irrespective of the kind of parsing algorithm used. Note that if we count the ambiguity of usages (syntactic and semantic differences) in addition to that of parts-of-speech, this sentence has 15 times ambiguity. The word **'consider'** has 3 different usages and the word **'process'** has 4 different usages as a noun in the dictionary of Mu project. Therefore, the ambiguity of this sentence is $3 \times (4 + 1) = 15$.

In a practical system, however, we encounter a lot of ambiguities. The reason is that a sentence in a practical application is longer than a sentence referred to in a model system. For example, a sentence which contains the word **'process'** in a real abstract is:

> **The authors have developed a technique to probe the dynamical details of the laser-induced solid-phase epitaxial (SPE) growth process in ion-implanted and deposited amorphous films and have extended these studies to a temperature regime which produces growth rates in excess of 10^6 times those measured by conventional methods of surface analysis.**

Table 4.1 shows the ambiguity of real sentences. This data is measured by about 3,000 sentences in abstracts of scientific papers in INSPEC database which is used as the English corpus to develop the translation system of the Mu project.

One of the most ambiguous sentences in the examples is:

> **Consequently, the following circuit was developed: it used two 400 V FETs (available in a wide range of current ratings) in cascade,**

to act as an 800 component which can be driven just like a single FET but which automatically splits the voltage load in such a way that neither FET 'sees' more than half the maximum voltage.

This sentence has 64 words (compound words) including punctuation marks[1] and about 3.58×10^9 times ambiguity, if we simply multiply the number of parts-of-speech of each word. The beginning part of the result of a morphological analysis is shown in figure 4.2 to illustrate how many ambiguities is in this sentence. For example, a simple word sequence **'a wide range'** has 8 times ambiguity, because each word has 2 alternative parts-of-speech. Note that even a very simple word **'a'** has 2 parts-of-speech, because **'a'** is sometimes used as a symbol for expressing a constant in the abstracts.

To solve this problem, the Mu project uses a lot of heuristic rules for the disambiguation. The heuristic rules are expressed as a combination of subgrammars and rewriting rules. Several heuristic rules are shown in figure 4.3 to 4.8. In the rest of this section, these are explained to show how to use GRADE grammar rules to disambiguate parts-of-speech.

Subgrammar to Control Disambiguation Process

Figure 4.3 is the subgrammar for the disambiguation of parts-of-speech, which is used in the Mu project. This subgrammar consists of 23 rewriting rules, which are described after the GRADE keyword 'rr_in_sg.'[2] Note that each rewriting rule calls other subgrammars in its substructure operation part and they also consists of many rewriting rules. Hence names of rewriting rules shown in the rest of this section do not appear in figure 4.3.

These rewriting rules are classified into 4 groups depending on the functions of the rules:

1. Word specific rules. (Group-0)

2. *Strong* heuristic rules referring to parts-of-speech. (Group-1)

[1]The number of words depends on the entries in the analysis dictionary. 'Maximum voltage', for example, has a single entry as a compound word in the dictionary of the Mu project.

[2]In GRADE grammar rules of the Mu project, an English part-of-speech is a value of the property name E_CATS, which is an abbreviation of 'English CATegory for analysiS.'

```
? <?> <ROOT>
|--ADV <consequently>
|--SYMB <,>
|--ART <the>
|--? <?> <PARA>
| |--N <following>
| |--ADJ <following>
| |--V <follow>
|--? <?> <PARA>
| |--N <circuit>
| |--V <circuit>
|--V <be> <USAGE_PARA>
| |--V <be>
| |--V <be>
|--V <develop> <USAGE_PARA>
| |--V <develop>
| |--V <develop>
|--SYMB <:>
|--PRON <it>
|--V <use>
|--CARD <two>
|--N <?>
| |--? <?> <MORCOMP>
| |--CARD <400>
| |--UNIT <V>
|--N <FET>
|--SYMB <(>
|--ADJ <available>
|--PREP <in>
|--? <?> <PARA>
| |--N <a>
| |--ART <a>
|--? <?> <PARA>
| |--ADJ <wide>
| |--ADV <wide>
|--? <?> <PARA>
| |--N <range>
| |--V <range>
```

Figure 4.2: Beginning Part of the Result of Morphological Analysis of Sample Sentence

```
DISAMBIGUATE_POS.sg;
  sg_mode;  order(2);
  rr_in_sg;
    DISAMB_PUT_PRE_CATS;
  /** WORD SPECIFIC RULES AND PATCH RULES ** GROUP-0 */
    DISAMB_PATCH;     DICCOMP_PATCH;     PARA_DICCOMP;     PARA_ALONE;
  /*** DISAMBIGUATE E_CATS, PART-1 **** GROUP-1 */
    DISAMB_CUT_1;     DISAMB_CUT_2;      DISAMB_CUT_POST; DISAMB_INDIV;
    DISAMB_ADV_LIKE;
  /********** CHANGE E_CATS *********** GROUP-2 */
    DISAMB_QUEST;     DISAMB_THATS;      DISAMB_VERBS;
  /*** DISAMBIGUATE E_CATS, PART-2 **** GROUP-3 */
    DISAMB_AUXV;      DISAMB_RELPRON;    VN_NOND;          LIKE_1;
    DISAMB_PARA;      DISAMB_PREPCONJ;   DISAMB_DOUBLE;
    DISAMB_TOP_OF_SENTENCE;              DISAMB_LAST_OF_SENTENCE;
    DISAMB_BY_NEIGHBOR;
end_sg.DISAMBIGUATE_POS;
```

Figure 4.3: Subgrammar for Disambiguation of Parts-of-Speech

```
VN_WHICH.rr;
  mi;     level(0,0); order(1); list;
  mc;     %(WHORD (PARA N V) X);
          WHORD.E_LEX = 'which';
          PARA.$SYS$ = 'PARA'; N,V : disorder;
          X.E_CATS =~ 'AUX'; X.E_CATS =~ 'V';
  cr;     %(WHORD V X);
          V.E_DISAMB_RULE <= 'VN_WHICH';
end_rr.VN_WHICH;
```

Figure 4.4: Word Specific Disambiguation Rule for the Word 'which'

3. Rules for handling specific words (e.g. 'that') and specific syntactic form (e.g. interrogative forms). (Group-2)

4. *Weak* heuristic rules referring to parts-of-speech. (Group-3)

These rules are basically arranged in the priority (reliability) of each rule. Therefore, these rewriting rules are applied to an input sentence sequentially in this order. This application order is specified in the subgrammar mode part (SG_MODE).

Word Specific Rule — 'which'

```
MAINING_1.RR;
  MI;  LEVEL(0,0); ORDER(1); LIST;
  MC;  %(VAUX ADV V);
       VAUX.E_LEX = 'be'|'by'; ADV : OPTIONAL;
       V.E_NONFINITE_FORM = 'ING'; V.E_DIS =~ 'V'; V.E_DISV = NIL;
  CR;  **;
       V.E_DIS <= 'V'; V.E_DISV <= T;
END_RR.MAINING_1;
```

Figure 4.5: Rule to Disambiguate 'ing' Formed Word Follows 'be' or 'by'

Some of the ambiguity of a word which follows the word **'which'** is resolved by the rewriting rule shown in figure 4.4.[3] This rule describes the heuristics:

> If 'which' is followed by a Noun, then a word which follows the Noun is usually a Verb (word sequence: 'which' Noun Verb). This means that if a word which follows 'which' is a Noun or Verb, it cannot be a Noun.

The tree structure pattern of this rule, which is described in the matching condition part (mc), expresses a word sequence:

1. A wh-word (WHORD) whose lexical unit (E_LEX) is 'which.'
2. An ambiguous word, which is expressed as 'PARA.SYS = 'PARA'',[4] whose part-of-speech is 'Noun' or 'Verb.'
3. A word whose part-of-speech is neither 'AUXiliary verb' nor 'Verb.' (e.g. X.E_CATS =~ 'AUX')

If the GRADE system finds this pattern, it chooses the 'Verb', and removes 'Noun' interpretation, which is expressed in the creation part (cr).

Rules to Disambiguate 'ing' Word Form

[3] In GRADE, grammar writers may use abbrevated keywords: mi (matching instruction), mc (matching condition), sso (substructure operation), cr (creation), etc.

[4] 'SYS' is a special property name and its value is usually set by the GRADE system. In this case, the value PARA means that this node is a *para special node*. See section 3.7.

```
NING_1.rr;
 mi;  level(0,0); order(1); list;
 mc;  %(A V C);                                  /* V(ING) => N */
      A.E_CATS = 'ADJ'|'ART'|'DET';
      V.E_NONFINITE_FORM = 'ING'; V.E_DIS =~ 'V'; V.E_DISV = NIL;
      C.E_CATS = 'PUNC'|'PREP'|'V'|'CONJ';
 cr;  **;
      V.E_CATS <= 'N'; V.E_VERB_RELATED <= '+';
      V.E_NUMBER_TYPE <= 'U';  V.E_DISAMB_RULE <= 'NING_1';
end_rr.NING_1;
```

Figure 4.6: Rule to Disambiguate 'ing' Word Form by Pattern

An 'ing' word form (verb) often causes a problem for the parsing, because it works as a noun, an adjective, and a predicate (verb). The rule[5] in figure 4.5 shows the heuristics:

An 'ing' word form which follows a 'be-verb' or 'by' is usually used as a verb.

An 'ing' word form is also disambiguated by the rule shown in figure 4.6. This rule disambiguates an 'ing' word form as Noun, if a word sequence matches the pattern:

$$\left|\begin{array}{c}\text{ADJ}\\ \text{ART}\\ \text{DET}\end{array}\right| + \text{V-ING} + \left|\begin{array}{c}\text{PREP}\\ \text{V}\\ \text{CONJ}\\ \text{PUNC}\end{array}\right| \Rightarrow \text{N}$$

Note that the rules in figure 4.5 and 4.6 do not transform a tree structure. This is expressed by the GRADE keyword '**' in the creation part (cr). Instead of that, these rules add several property values to nodes (e.g. 'V.E_CATS <= 'N''). This is another feature of the GRADE system.

'Negative' Rules

The rules in figure 4.4, 4.5, and 4.6 are 'positive' rules, which select the correct part-of-speech of a word. On the other hand, there are 'negative' rules, which remove impossible parts-of-speech in a sentence-level context. The rule in figure 4.7 is one of

[5] An 'ing' word form is treated as an inflected verb (its part-of-speech is V, and its nonfinite form type is ING) at the morphological analysis phase in the Mu project. This is a tentative interpretation during the phases before the disambiguation to suppress unnecessary combinatorial explosion. The correct part-of-speech for an 'ing' formed word is determined at this phase.

```
DISAMB_CUT_1.rr;
 mi;  level(0,0); order(2,noskip); list;
 vi;  @XX;
 mc;  %(A (PARA #1 B #2));
      A.E_CATS = 'ART'|'DET'|'PREP';
      PARA.$SYS$ = 'PARA'; B.E_CATS = 'V'|'ADV'; PARA.E_DIS =~ 'U';
 sso; PARA.E_DIS <= 'U';
      @XX <= call-sg(DISAMB_CUT_1 %(A PARA) list);
 cr;  %(@XX);
end_rr.DISAMB_CUT_1;
```

Figure 4.7: Rule to Remove 'Impossible Parts-of-Speech'

```
LIKE_1.rr;
  mi;    level(0,0); order(1); tree;
  mc;    %((? A (PARA #1 V #2) #3));
         PARA.$SYS$ = 'PARA'; V.E_LEX = 'like';
         A : length(0,200,%(A1));
         A1.E_CATS =~ 'PRON';
         A1.E_CATS =~ 'N' or (A1.E_SEM ain '(OH OB)') = nil;
  cr;    %((? A (PARA #1 #2) #3));
         PARA.E_DISAMB_RULE <= 'LIKE_1';
end_rr.LIKE_1;
```

Figure 4.8: Word Specific Rule to Disambiguate the Part-of-Speech of 'like'

'negative rules' called by the subgrammar DISAMBIGUATE_POS in figure 4.3. This rule expresses heuristics:

A word which follows 'ARTicle' or 'DETerminer' is not 'Verb' or 'ADVerb.'

Note that the real disambiguation rules are described as another subgrammar whose name is 'DISAMB_CUT_1.' The rewriting rule in figure 4.7 calls it in the sub-structure operation part:

```
sso; @XX <= call-sg(DISAMB_CUT_1 %(A PARA) list;
```

Word Specific Rule — 'like'

There are rules which refer to semantic markers of a noun. Figure 4.8 shows the word specific rule for 'like':

> If 'like' does not follow a PRONoun or a Noun whose semantic markers do not include (ain) Human (OH) or Animal (OB), 'like' is rarely a Verb.

In the phase of an application of this rule, the system does not know a global structure, such as Noun Phrase. Therefore, this rule scans an input word sequence with the LENGTH option of a GRADE grammar rule to confirm that there is no 'PRONoun' or 'Noun' whose semantic markers include 'human' or 'animal.'

Rules to Fix Wrong Decision

Heuristic rules discussed in this section sometimes make a wrong decision. However, the grammar of the Mu project has subgrammars to fix the wrong decision, when the system finds an inconsistency in the later analysis phases [Kume87].

4.2.2 Problems Caused by Constructions of Sentences

Constructions of sentences, such as itemized forms, cause serious problem of MT parsers. There are many such exceptional constructions in written texts, especially in the abstracts of scientific and technological papers which the Mu project aims to translate.[6] The following is a typical example:

> **Four major factors affect the cost of ownership: 1) purchase price, 2) investment tax credits, 3) cost of maintenance and repairs and 4) efficiency costs.**

This type of construction can be handled by the rules of the context free grammar (CFG) shown in figure 4.9. However, if rewriting rules to handle such sentence constructions are added to the analysis grammar, the following problems would arise:

1. Deterioration of analysis efficiency: Rewriting rules which need not be referred to in a structural analysis will increase. Since CFG parsers cannot distinguish the rewriting rules for the structural analysis from the rewriting rules for the typographical sentence construction analysis, the increase in the total number of rules reduces the efficiency of the analysis.

[6]The English corpus of the Mu project consists of the abstracts extracted from the INSPEC database without any pre-editing.

```
S   →  S' ':' IF '.' IF IFL 'and' IFE
IFL →  IFE IFL IFE ',' IFL
IFE →  IN NP IN NUMBER ')'
NP  →  DET N                          rules for itemized forms
       ...
S   →  S' '.'
S'  →  NP VP                          rules for structural analysis
       ...
```

S:	Sentence,	IF:	Itemized Form,
IFL:	List of Itemized Elements,	IFE:	Element of Itemized Form,
IN:	Item Number,	NP:	Noun Phrase

Figure 4.9: Context Free Rules for Processing Itemized Forms

2. The loss of useful heuristics for correct analysis: For example, a heuristic 'each item in an itemized form can be analyzed independently' is useful for the analysis grammar. To use such heuristics, the recognition of global sentence structures should precede the detailed structural analysis. It, however, cannot be utilized effectively in an analysis grammar based on CFG.

3. Deterioration of the maintainability of the analysis grammar: It becomes difficult to distinguish rules concerned with particular text types (i.e., abstracts of scientific papers, articles of newspapers, etc.) from rules concerned with more general linguistic phenomena.

In contrast to such an approach, tree-to-tree rewriting rules and using subgrammar-networks to control the parsing, which are features of GRADE, allow the MT parser to analyze such sentential forms without the deterioration of efficiency and maintainability. The analysis procedure of, for example, an itemized form is the following:

1. First, the fragments of an input sentence are separated from each other by a tree structure pattern such as that shown in figure 4.10.

2. Each fragment is analyzed independently by the main-analysis subgrammar.

```
...     ':'    [ NUM ')' ... ',']+   NUM ')' ...   'and'   NUM ')' ...     '.'
#0   COLON          IFL1               IFE2         AND       IFE3       PRERIOD
```

(a) Pattern to extract fragments from a sentence

```
%( #0 COLON IFL1 IFE2 AND IFE3 PERIOD);
  IFL1: length(1,10,%(NUM1 RIGHT_PAREN1 #1 COMMA));
  IFE2: %(NUM2 RIGHT_PAREN2 #2);
  IFE3: %(NUM3 RIGHT_PAREN3 #3);
```

(b) GRADE pattern

Figure 4.10: Example of the Pattern to Extract Fragments from Itemized Form

3. Finally, fragmental results are integrated into the whole analysis result by the post-analysis subgrammar.

In this example, the first phase separates the input sentence of the example into the core part of the sentence and the itemized parts as follows:

Core part (#0):	Four major factors affect the cost of ownership
Itemized Part (IFL1):	1) purchase price, 2) investment tax credits,
Itemized Part (IFL2):	3) cost of maintenance and repairs
Itemized Part (IFL3):	4) efficiency costs.

The second phase analyzes these fragments independently as a sentence and noun phrases. Then the third phase integrates the result.

In this way, the grammar rules for analyzing sentence constructions are placed in the pre-analysis subgrammar and are separated from the main-analysis subgrammar where the syntactic and semantic analysis of the input sentences is performed. This separation avoids the degradation of both analysis efficiency and the grammar's maintainability — both serious problems in parsers based on CFG.

4.2.3 Problems Caused by Structural Ambiguity

Since the MT parser has to analyze fairly long sentences containing various sorts of constructions, such as long conjuncted phrases, appositions, ellipses, etc., it has to use heuristic methods for the disambiguation to determine the priorities of the possible syntactic and semantic interpretations produced for an input sentence.

The analysis methods based on CFG usually handle each interpretation independently. Therefore, the following two methods are typically useful for determining the priorities of such interpretations:

1. First, obtaining all possible interpretations from the input sentence and then comparing them. Note that such rules have to compare different interpretations (tree structures) and cannot be CFG rules.

2. Heuristically or statistically adjusting the order of application of CFG rewriting rules, for example, to obtain just one interpretation from the input sentence and to ignore others [Nagao82b].

Unfortunately, the first method lowers the parsing efficiency because of the inherent 'combinatorial explosion'. It often exhausts the limited computer time and memory and still gives no interpretation. The second method has difficulties in maintaining the priority ordering of rewriting rules adequately. This difficulty increases with any increase of rules in the whole grammar.

However, many kinds of ambiguity can be solved by adopting the controlling mechanism provided by GRADE's subgrammar ('procedural analysis' method [Tsujii84]). In this section, we will focus on how to disambiguate the interpretation of functional words like 'after', 'as', and 'for' which can be used both as prepositions and as (sentential) conjunctions. These functional words bring about the two problems as follows:

1. Processing efficiency; there are certain kinds of ambiguities which may be solved automatically, when we use a CFG based parser augmented by simple semantic checking. For example, 'as' and 'after' in the following sentences are used as prepositions and are not used as conjunctions:

   Remarkably, the printed board can be executed as a one-sided or a double-sided unit. — (2)

   The solderability of reflowed tin and 40 percent lead coated copper has been examined after thermal aging designed to

include extensive copper-tin intermetallic compound growth.
— (3)

However, a lot of computer time and memory are necessary, because the number of possible partial interpretations increases combinatorially. For the correct interpretation of (3), we need such a semantic constraint as the agent of 'to **design**' should be a *human*. However, such a semantic constraint is not a real constraint, but a preference.

2. Ambiguous interpretation; for the correct disambiguation, a complete semantic and contextual analysis is necessary. The word '**for**' in the following sentence is ambiguous.

   Many opportunities occur <u>for</u> contractors to obtain electrical maintenance work in factories. — (4)

   This sentence has two possible syntactic structures as follows:

   Many opportunities occur [$_S$ for contractors [$_{VP}$ to obtain [$_{NP}$ electrical maintenance work in factories.]]] — (4.1)

   Many opportunities occur for [$_S$ [$_{NP}$ contractors to obtain electrical maintenance] [$_{VP}$ work in factories.]] — (4.2)[7]

   The dominant reading is (4.1): 'contractors obtain a work', but we cannot reject (4.2): 'contractors work in factories.'

Both these problems must be solved by an MT parser. The efficiency problem can be solved by adopting the 'three stage procedural analysis' as follows [Yamamoto86]:

- Step 1. Disambiguation by simple but reliable cues. Rules for disambiguating parts-of-speech are applied. For example, it is tentatively determined that 'as' in (2) is a preposition by simple but rather reliable cues such as:

[7]Usually, a comma or a semicolon is inserted between 'obtain' and 'for.'

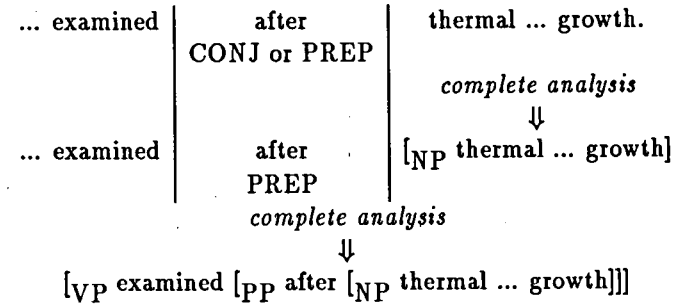


Figure 4.11: Example of Disambiguation by Partial Analysis

If there are no verbs after the ambiguous function word, its part-of-speech is a preposition.

Such a rule is easily expressed by using the flexible pattern matching functions of GRADE.

- Step 2. Disambiguation based on intermediate analysis result. **'After'** in (3) cannot be disambiguated in step 1. The ambiguity is solved in this step by the following rule:

  If the word sequence after the ambiguous function word is to be analyzed as a sentence, the word is a conjunction. If the phrase is a noun-phrase, then the word is a preposition.

  For (3), first, the word sequence **'thermal ... growth'** after **'after'** is extracted and analyzed completely. The grammar for complete analysis, written as a subgrammar network, is called from this rule. For this sentence, the word **'after'** is determined as a preposition, because the word sequence following **'after'** is analyzed as a noun-phrase (see figure 4.11).

  This kind of top-down processing is easily realized by the pattern matching functions and the invocation of subgrammar-networks.

- Step 3. Complete Analysis. After step 2, each word (or phrase) is no longer (syntactically) ambiguous. The complete analysis of the sentence therefore becomes

straightforward. However, the determinations of parts-of-speech in step 1 and step 2 are tentative, and it sometimes happens that complete analyses cannot be obtained because wrong decisions have been made. In such cases, these tentative decisions may be changed by step 3.

During step 1 and step 2's construction of tentative interpretations, various sorts of heuristic rules are applied; these are ordered according to their relative 'strengths'. The part-of-speech interpretation of 'for' in (4), for example, shows that a heuristic rule based on surface syntactic cues such as

> If there is a word sequence '... for NP to verb ...', the 'for' is a preposition which marks the subject of the infinitive clause.

is useful. This kind of heuristic rules is useful for choosing the most feasible interpretations, even if there are several syntactically and semantically possible interpretations.

The '3 stage procedural analysis', in which bottom-up processes (step 1) and top-down processes (step 2) are carefully combined, can be implemented straightforwardly by utilizing the rich control schemes provided in GRADE.

4.2.4 Flexibility of Transfer Phase

The transfer phase is another key point of a machine translation system. In the transfer phase, the machine translation system needs to execute two different tasks to obtain high quality translations [Nagao85c]:

1. Word selection: To select a translated word properly, the system should have a function to refer to a local context of the source language word.

2. Adjustment of a syntactic structure: There are lot of syntactic and semantic differences between Japanese and English, for example. To obtain a *natural* translation, we need to transform an input structure, which is closely related to the source language, into a structure suitable for the target language.

These tasks in the transfer system of the Mu project are effectively expressed with the functions of GRADE:

NYURYOKU-SURU (to input)

TAIWA-KEISHIKI (interactive form)
Deep-Case: YOTAI
Surface-Case: DE

Figure 4.12: Result of Japanese Analysis

- Flexible pattern matching function: This is used to find a pattern for the word selection and the structure adjustment.
- Dictionary rules: Most of rules to obtain a *natural* translation are depend on specific words. These are expressed as dictionary rules.

In the rest of this section, several rules used in the Japanese-into-English transfer system of the Mu project are explained. The outline of the transfer is discussed in section 4.3.2.

Word Selection of Noun

Let us consider a Japanese sentence:

TAIWA-KEISHIKI DE NYURYOKU-SURU (to input interactively)

The result of the analysis of this sentence (tree structure) is shown in figure 4.12.[8] The verb 'NYURYOKU-SURU (to input)' governs the noun 'TAIWA-KEISHIKI (interactive form)' as a 'YOUTAI-case (manner).' In general, a noun phrase governed by a verb in Japanese is translated into a prepositional phrase in English, except a subject and an object. Therefore, the direct transfer for this sentence becomes the tree structure shown in figure 4.13. This type of transformation is handled by general rules of the transfer system and the final translation becomes a sentence:

to input with interactive form.

However, this is not a *natural* translation. To solve this problem, the dictionary rules are used in the transfer phase of the Mu project. In this case, the dictionary rule for the

[8]In the examples of rules and tree structures in this chapter, Japanese words (e.g. 'TAIWA-KEISHIKI') is expressed in Romaji for clearness of the discussion. However, in the real system, all Japanese words is expressed with Kana and Kanji combination (Kanji-code) directly.

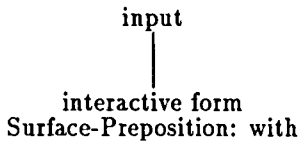


Figure 4.13: Result of Transfer by General Rules

```
taiwa-keishiki_N_TRANS_B1.RR;
 MI;  LEVEL(0,0); ORDER(1); TREE;
 MC;  %((VA CASE));
   CASE: %( (X1 $) X2); X1,X2:DISORDER_SKIP( %( ? ) );
   X1.J_DEEP_CASE OEQ 'youtai'; X1.J_SURFACEP_CASE OEQ 'de';
   X1.J_LEX  =  'taiwa-keishiki'; VA.J_CAT = 'doushi';
 CR;  %((VA  CASE ));
   X1.E_LEX = 'interactively'; X1.E_CAT = 'ADV'
END_RR.taiwa-keishiki_N_TRANS_B1;
```

Figure 4.14: Dictionary Rule for the Noun 'TAIWA-KEISHIKI'

noun 'TAIWA-KEISHIKI', shown in figure 4.14, is applied to the result of the analysis in figure 4.12. This rule generates the transferred English structure in figure 4.15. Finally the translation becomes a sentence:

to input interactively.

Note that if the noun 'TAIWA-KEISHIKI' is modified by an adjective phrase, this dictionary rule fails its application and does not execute this transformation. For example, this rule does not succeed its application for the sentence:

KIKAI TONO TAIWA-KEISHIKI DE NYURYOKU-SURU (to input by an interactive mode with a machine.)

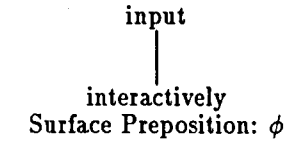


Figure 4.15: Result of Transfer by Dictionary Rule

The pattern '(X1 $)' in figure 4.14 expresses this constraint, because '$' means a *null* tree in GRADE grammar rules.

Most of such word specific information is described as *general* forms in the transfer dictionary. They are transformed into the GRADE grammar rules by the dictionary sub-system of GRADE, which is discussed in chapter 5.

There are, however, several word specific rules difficult to express as *general* dictionary forms. For example, the Japanese suffix-word 'SEI' has several English translations depend on attached nouns:

KINZOKU (metal)	SEI	⇒	made of metal
BEIKOKU (U.S.A)	SEI	⇒	made in U.S.A
IBM	SEI	⇒	made by IBM

Such word specific rules are written by grammar writers directly with GRADE grammar rules. In the transfer phase, however, GRADE system treats them as the same as the general dictionary rules.

Adjustments of Syntactic Structure

Embedded sentences in Japanese are translated into various English structures: relative clause, adverbial clause, infinitive form, prepositional phrase (e.g. for ~ing, of ~ing), that-clause and so on. Figure 4.16 shows the example (type 2 [Nagao85c]) of the structural adjustments for the relative clauses:

HANKEI GA OOKII EN (circle whose radius is large)
⇒ circle with (a/the) large radius

Rules for handling relative clauses are expressed as subgrammars. Figure 4.17 is one of the rewriting rules in the subgrammars for the relative clause transfer. This rule finds the phrase governed by 'ADJective', which is expressed by the condition:

```
VADJ.E_CAT = 'ADJ'.
```

If the phrase is not negated:

```
VADJ.E_S_NEGATIVITY =~ 'T',
```

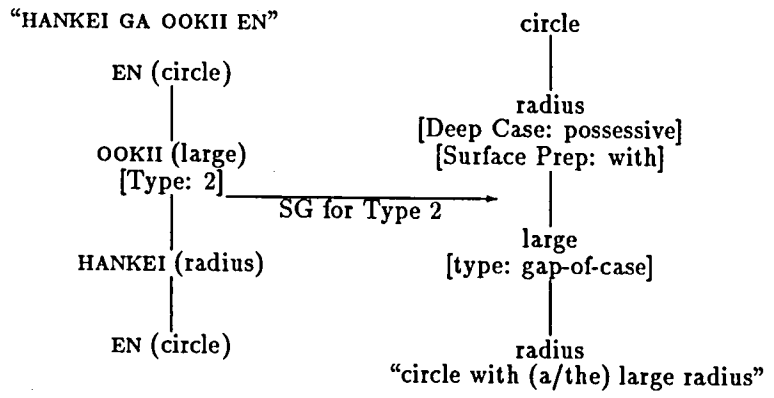


Figure 4.16: Structural Adjustment for Relative Clauses Type 2

the noun is moved to the top of the phrase. This is expressed in the creation part by the pattern:

```
%(( J-NX (VADJ J-F) )).
```

4.3 Grammars Developed in Mu Project

In the Mu project, two translation systems have been developed: the Japanese into English system [Nagao84] [Nagao86a] [Tsujii84] and the English into Japanese system [Kume87] [Tsujii85b] [Yamamoto86]. In this section, the outline of the Japanese into English system is described, as an example of the usage of the GRADE system.

4.3.1 Japanese Analysis

Flow of Analysis: Procedural Grammar

The analysis grammar to parse Japanese sentences in the Mu project is divided into several sub-components depend on the phases of the analysis procedure. The whole grammar is expressed with the subgrammar network and each component is expressed

```
GINP_REVERSE_CHECK.rr;
  mi; level(0,0); order(1); tree;
  mc;  %(( VADJ CASE ));
    VADJ.E_CAT = 'ADJ';  VADJ.E_S_NEGATIVITY =~ 'T';
    CASE: %( J-N1 J-F );
    J-N1, J-F: disorder_skip( %( X ) );
    J-N1.J_DEEP_CASE = 'SUBJECT'; J-N1.J_MODIFIED = 'T';
    J-F: optional;
    (X.J_TYPE = 'GAP') or (X.$SYS$ = 'DICCOMP');
  cr;  %(( J-NX (VADJ J-F) ));
    J-NX.* <= J-N1.*; J-NX.E_DEEP_CASE <= 'ETC';
    J-NX.E_SURFACE_CASE <= 'A_COMPO';
    J-NX.E_PREP_LEX <= 'with';
end_rr.GINP_REVERSE_CHECK;
```

Figure 4.17: Rewriting Rule for Relative Clauses Type 2

with subgrammars and subgrammar networks. Following is the flow of the Japanese analysis part:

1. Morphological analysis. For the efficiency of the execution, this phase is implemented in LISP directly, instead of using the GRADE system.

2. Disambiguation of words that can be assigned to multiple parts-of-speech.

3. Handling of compound words.

4. Determination of the scope of sentences ending with 'Renyo' form verbs.[9]

5. Determination of the scope of noun phrases linked by conjunctions.

6. Determination of the scope of noun phrases and determination the relationships among words in noun phrases. This phase also includes determination of embedded sentence types.

7. Determination of the relationships among words in simple sentences. This is based on the 'case frame' analysis. This phase includes interpretation of the meaning of optional cases.

[9]'Renyo' form verbs express a compound sentence in Japanese. 'Renyo' form verbs cause a lot of scope ambiguities, like the English word 'and' and prepositional phrases.

8. Determination of the role of noun phrases followed by the post positional particle 'HA', which has several interpretations: subject marker, topic marker, and so on.

9. Determination of the semantic relationships among sentences and among words in noun phrases.

10. Tense and aspect processing.

11. Ellipsis processing.

12. Conversion of the phrase structure (the output of phase 11) into a dependency structure.

The analysis grammar is based on the 'procedural grammar' approach [Tsujii84]. This aims at the determination of a single result of the analysis at each phase. In the following of this section, typical phases of the Japanese analysis are explained.

Post Processing of Morphological Analysis

An input sentence to the Mu project system is first morphologically analyzed and separated into each word (phase 1). Figure 4.18 shows an example of the output from the morphological analysis program for a sentence:

DENKI-KEISOKU-HOU ,	DEETA-SHORI ,	JIDOU-KA-KIKI
(electrical measurement)	(data processing)	(automated equipment)

NO	SHINPO DE	JIDOUKA-SEN GA	ZOUKA-SHI-TA .
(of)	(with advance)	(automated ship)	(increased)

This sentence is mainly used for explanation in this section.

In the tree structure of figure 4.18, the compound word 'DENKI-KEISOKU-HOU', for example, is expressed as a single entry word. The internal structure of this word is also expressed with sub-trees (e.g. 'DENKI'). The grammar rules refer to such a compound word as a single word, in general. However, they refer to its internal structure, if the grammar rules need to do so.

The subgrammars for the disambiguation of parts-of-speech and local analysis of the compound words, which are not stored as single entry words in the dictionary, are first

```
?  <?>
|--N  <denki-keisoku-hou>
|  |--?  <?>
|     |--?  <denki>
|     |--?  <keisoku>
|     |--?  <hou>
|--PUNC  <TOU_TEN>
|--N  <deeta-shori>
|  |--?  <?>
|     |--?  <deeta>
|     |--?  <shori>
|--PUNC  <TOU_TEN>
|--N  <jidou-ka-kiki>
|  |--?  <?>
|     |--?  <jidou>
|     |--?  <ka>
|     |--?  <kiki>
|--BKK  <no>
|--N  <shinpo>
|--?  <?>
|  |--BKK  <de>  <DE>  <SUBJECT...UO>
|  |--A  <da>
|--N  <jidou-ka-sen>
|  |--?  <?>
|     |--?  <jidou>
|     |--?  <ka>
|     |--?  <sen>
|--BKK  <ga>  <GA>  <SUBJECT...UO>
|--DSI  <zoukasuru>
|--A  <ta>
|--PUNC  <KU_TEN>
```

Figure 4.18: Output of Japanese Morphological Analysis

```
J_DISAMB_DE_PUNC_BKK_2.rr;
  mc; %( (?  NOUN_PHRASE (P BKK A) PUNC #999) );
    NOUN_PHRASE:length(1,100,%(X)) ;
    (X.J_CAT = 'N' | 'Z' | 'R')                                    or
    ((X.J_CAT = 'ADJ') and (X.J_INF = 'RENTAI'))                   or
    ((X.J_CAT = 'ADJ') and (X.J_TAIL = nil))                       or
    ((X.J_CAT = 'BKK') and (X.J_INF = 'RENTAI'))                   or
    ((X.J_CAT = 'C') and (X.J_C = 'SKU'))                          or
    ((X.J_CAT = 'PUNC') and (X.J_PUNC = 'TOU_TEN' | 'NAKAGURO')) ;
    P.J_AMB = 'T'; BKK,A : disorder;
    A.J_LEX = 'da' | 'dearu'; PUNC.J_PUNC = 'TOU_TEN';
  cr; %( (?  NOUN_PHRASE BKK PUNC #999) );
    BKK.J_DISAMB <= 'J_DISAMB_DE_PUNC_BKK_2';
end_rr.J_DISAMB_DE_PUNC_BKK_2;
```

Figure 4.19: Rewriting Rule for Disambiguation of 'DE-DEARU' (1)

applied to this tree structure (phase 2 and 3). These phases remove local ambiguities in an input sentence with heuristic rules.

In the example of figure 4.18, there is ambiguity of parts-of-speech for the word 'DE.' In the surface level, 'DE' has two possibilities: a KAKU-JOSHI (case marker whose typical meaning is 'with'; in the rules, expressed as 'BKK'), and an inflected form of the auxiliary verb ('A') 'DEARU (be)', which can be placed after a noun. Heuristic rules are used to determines its *feasible* part-of-speech. These rules are similar to the heuristic rules for English parts-of-speech disambiguation discussed in section 4.2.1. Two of the rewriting rules for the disambiguation for 'DE' are shown in figure 4.19 and 4.20. In this example, the system chooses 'KAKU-JOSHI' (BKK) as the part-of-speech of 'DE.'

The rewriting rule in figure 4.19 finds a pattern:

NP NP ... DE , ...

in the surface level of an input sentence. If this pattern is found, this rule chooses BKK (case marker) as a part-of-speech of 'DE.' The pattern expressed in the matching condition part (mc) specifies:

> 'DE' follows Noun Phases which only consist of Nouns, ADJectives whose inflected forms are 'RENTAI (attributive)', attributive case markers (BKK), and so on.

```
J_DISAMB_DE_PUNC_A_7.rr;
  mc; %( (?  #997 BFK #998 (P BKK1 A1) PUNC #999) );
    BFK.J_LEX =  'ha' | 'mo' ;
    P.J_AMB = 'T'; BKK1,A1 : disorder;
    A1.J_LEX = 'da' | 'dearu'; PUNC.J_PUNC = 'TOU_TEN';
  cr; %( (?  #997 BFK #998 A1 PUNC #999) );
    A1.J_DISAMB <= 'J_DISAMB_DE_PUNC_A_7';
end_rr.J_DISAMB_DE_PUNC_A_7;
```

Figure 4.20: Rewriting Rule for Disambiguation of 'DE-DEARU' (2)

This expresses the heuristics:

> If there is no word before 'DE' which should be governed by a predicate, 'DE' is usually a 'case marker.'

On the other hand, the rewriting rule in figure 4.20 finds the pattern:

> ... HA ... DE , ...

This rule chooses the auxiliary interpretation 'DEARU' for 'DE' in this pattern, because 'HA' often co-occurred with 'DEARU.' Note that this rule expresses relatively *weak* heuristics. Therefore, this rule is only tried after several *stronger* heuristic rules are applied to an input sentence. The subgrammar gives this precedence to the analysis grammar.

By the rules in the phase 2 and 3, the tree structure in figure 4.21 is obtained. This tree shows that there is no part-of-speech ambiguities.

Sentential Conjunction

There are many compound sentences in the abstracts of technical papers. In Japanese, the construction of a compound sentence is called 'RENYOU-CHUSHI.' 'RENYO' form verbs cause a lot of scope ambiguities. Since 'RENYO' form verbs govern adverbial phrases, they cause an ambiguity problem like the PP-attachment in English. At the same time, 'RENYO' form verbs can modify a noun phrase and make an embedded sentence. For example, a simple word sequence:

```
? <?>
|--N  <denki-keisoku-hou>
|--PUNC  <TOU_TEN>
|--N  <deeta-shori>
|--PUNC  <TOU_TEN>
|--N  <jidou-ka-kiki>
|--BKK  <no>  <NO>
|--N  <shinpo>
|--BKK  <de>  <DE>  <SUBJECT...UO>
|--N  <jidou-ka-sen>
|--BKK  <ga>  <GA>  <SUBJECT...UO>
|--DSI  <zoukasuru>
|--A  <ta>
|--PUNC  <KU_TEN>
```

Figure 4.21: Result of Disambiguation of 'DE-DEARU'

'RENYO' V1 NP V2.

has two possibilities:

$[_S\ [_S$ 'RENYO'$]\ [_S\ [_{NP}$ V1 NP$]$ V2$]]$
(V1 modifies NP, and 'RENYO' and V2 are coordinated.)

and

$[_S\ [_{NP}\ [_{VP}$ 'RENYO' V1$]$ NP$]$ V2$]$
('RENYO' and V1 are modify NP)

In a real sentence, this scope ambiguity and the PP-attachment ambiguity are combined and they cause the combinatorial explosion, if we use a simple parsing algorithm.

To avoid the combinatorial explosion, this construction is handled at phase 4 in the analysis grammar of the Mu project. The basic idea to solve this problem is the same as the one discussed in section 4.2.2, which handles sentential constructions such as itemized forms.

Figure 4.22 shows one of the rewriting rules used in this phase. This rule finds a 'RENYO' form verb and the same non-RENYO form verb which is a 'volitional verb.' This condition is expressed in the matching condition part with

'RW1.J_RT = 'WV' | 'SDV'', and 'RWV1.J_LEX = WV.J_LEX.'

```
J_RWV_DIVISION_1.rr;
  mi; level(0,0); order(3);
  mc; %( (S1 #1 RWV1 #2 WV #3) );
    RWV1.J_CAT='VP'; RWV1.J_RT_INF='RENYOU'; RWV1.J_RT='WV'|'SDV';
    RWV1.J_RT_PASS=~'T';
    WV.J_CAT='VP'; WV.J_RT_INF=~'RENYOU'; WV.J_RT='WV'|'SDV';
    RWV1.J_LEX=WV.J_LEX;
  cr; %( (S1 (S2 (S3 #1 RWV1) (S4 #2 WV) ) #3) );
    S2.J_PARA<='T'; S2.J_INF<=WV.J_INF; RWV1.J_RT_PASS<='T';
    S3.J_RT_SS<='T'; S3.J_SENTENCE_END<='RENYOU';
  end_rr.J_RWV_DIVISION_1;
```

Figure 4.22: Rewriting Rule for 'RENYO' Form Verbs

If such a pattern is found, the system decides that these verbs make a coordinate phrase.

Analysis of Noun Phrases

Noun phrases are analyzed in several phases of the analysis grammar: phase 3, 5 and 6. The most complex part is determination of the scope of noun phrases linked by a conjunction (phase 5). Many heuristic rules are used in this phase and they are ordered in the subgrammars according to their priorities. One of the subgrammar for the analysis of coordinate noun phrase is shown in figure 4.23. In this subgrammar, the rules using '· (centered dot)' as a clue (e.g. CENTERED_DOT1) are first applied to the input sentence. After that, the rules finding the pattern 'TO ... TO NO' (e.g. TONO1) are applied, because the former rules are more reliable.

The heuristic rules use complex patterns to determine the scope of noun phrases. Figure 4.24 shows the rewriting rule in the subgrammar for the analysis of coordinate noun phrases. This rule finds the pattern: '... TO ... TO NO ....' If this pattern is found in an input sentence, this rule puts the words between 'TO's together, because these words should become a single noun phrase. Note that the Japanese word 'TO' does not always work as a conjunction, but 'TO' works as a case marker governed by a verb whose meaning is 'with.' Therefore, this rule uses a constraint 'LTO1.PSYM = '+'' in the matching condition part, which means that the word 'TO' is used as a conjunction. This mark (PSYM) is set by other rules before the application of this rewriting rule.

```
RANK_A.sg;
    rr_in_sg;
      TOP_NODE; /*PRE_PROCESS*/
      SINGLE_N;
      CENTERED_DOT1; CENTERED_DOT2; CENTERED_DOT3;
      CREATE_MORCOMP;
      SYMBOL1; SYMBOL2; SYMBOL3;
      RESTORE_MORCOMP;
      TONO1; TONO2; RCAND;
      SNG_N_LEFT; SNG_N_RIGHT;
      RENTAI_ISOLATED1; RENTAI_ISOLATED2;
end_sg.RANK_A;
```

Figure 4.23: Subgrammar for Analysis of Coordinate Noun Phrases

```
TONO1.rr;
    mi; level(0,0); list;
    mc; %( LTO1 LNON_TO (LTO3 (LMORCOMP #1 LTO4 #2)) );
         LTO1.R_EL = ~'+'; LTO1.PSYM = '+'; LTO1.J_LEX = 'to';
         LNON_TO: any( ~%( LTO2 ) );
           LTO2.PSYM = '+'; LTO2.J_LEX = 'to';
         LTO3.TONO = '+'; LTO3.J_LEX = 'to'; LTO4.J_LEX = 'to';
    cr; %( LTO1 (NP LNON_TO) #1 #2 );
         LTO1.R_EL <= '+';
end_rr.TONO1;
```

Figure 4.24: Rewriting Rule for Analysis of Coordinate Noun Phrases

The other problem in the noun phrase analysis is caused by the word 'NO' (basic meaning is 'of'). The word sequence 'NP1 NO NP2 NO NP3' has two possibilities:

$[_{NP}\ [_{NP}$ NP1 NO NP2] NO NP3]

and

$[_{NP}$ NP1 NO $[_{NP}$ NP2 NO NP3]]

The first one is a default interpretation. However, the second one is correct in some cases. Therefore, a heuristic rule is useful:

> If the head noun of NP1 is an 'adverbial noun'[10] related to 'time', and if the head noun of NP3 is a 'verbal noun'[11], then NP1 modifies NP3, instead of NP2.

This rule is applicable to a sentence:

SAKUNEN	NO	KYO-DAI	NO	HOUKOKU
(last year)	(of)	(Kyoto Univ.)	of	(report)

(a 'last year' report from Kyoto Univ.)

This heuristic is described by the rewriting rule in figure 4.25. 'MSA' is a sub-category of noun for 'verbal noun' and 'MFK' is for 'adverbial noun' in the Mu project. 'TT', for example, is a semantic marker for a 'time' related noun. Therefore,

'R.J_N = 'MFK'' and '(R.J_SEM ain '(TT TP TD TA TX)')'

express the constraint for NP1.

Applying the phase 3, 5 and 6, the tree structure shown in figure 4.26 is obtained for the sample sentence.

Case Frame Analysis

The analysis of simple sentences (sentences which contain one verb) in phase 7 is divided into 3 steps:

[10] A noun which works as an adverb in some case, like a 'time word' (e.g. today) in English.
[11] A noun which has a verbal derivation.

```
J_RENTAI_2.rr;
 mc; %((? #1 R #2 X1 #3));
   X1.J_CAT = 'N' | 'NP'; X1.J_N = 'MSA' ;
   R.J_N = 'MFK' ; (R.J_SEM ain '(TT TP TD TA TX)') = ~nil ;
   R.J_PASS=~'YES';
 cr; %((? #1 (NP11 R #2 X1) #3));
   R.J_PASS<='YES'; NP11.*<=X1.*; NP11.J_CAT <= 'NP' ;
end_rr.J_RENTAI_2;
```

Figure 4.25: Rewriting Rule for Analysis of Noun Phrases

```
?  <?>
|--NP  <shinpo>
|  |--ADJP  <?>
|  |  |--NP  <jidou-ka-kiki>  <>  <GOV>
|  |  |  |--N  <denki-keisoku-hou>
|  |  |  |--N  <deeta-shori>
|  |  |  |--N  <jidou-ka-kiki>  <>  <GOV>
|  |  |--BKK  <no>  <NO>
|  |--N  <shinpo>  <>  <GOV>
|--BKK  <de>  <DE>  <SUBJECT...UO>
|--N  <jidou-ka-sen>
|--BKK  <ga>  <GA>  <SUBJECT...UO>
|--VP  <zoukasuru>
|  |--DSI  <zoukasuru>  <>  <GOV>
|  |--A  <ta>
|--PUNC  <KU_TEN>
```

Figure 4.26: Result of Analysis of Noun Phrases

1. Extraction of a sentential part from an input sentence.

2. Determination of the scope of noun phrases governed by a verb and determination of semantic relations between noun phrase and verb (mapping from surface cases into deep cases). This step uses the 'para special node' to gather all possible interpretations.

3. Selection of the *most feasible* interpretation using 'heuristic evaluation functions.'

Step 2 applies a subgrammar which contains several rewriting rules (dictionary rules written in the verb dictionary) to an input sentence. Each rewriting rule maps surface cases (usually marked by case markers: GA, WO, NI, KARA, and so on) into deep cases (*subject, object, recipient, origin*, and so on). Then the subgrammar makes one tree structure whose top node is the para special node.

Figure 4.27 shows the result of this step. This tree is an intermediate tree structure for the analysis of a simple sentence:

> HYOU NI HOUTEI-SHIKI KARA MOTOME-TA (IPPAN-KAI WO SHIMESU.)
> (to show (a/the) general solution derived from (a/the) equation in (a/the) table.)

There are two mapping between surface cases and deep cases for the verb 'MOTOME-TA':

1. NI ⇒ *object?*, KARA ⇒ *origin*, WO ⇒ *object*

2. NI ⇒ *recipient*, KARA ⇒ *subject?*, WO ⇒ *object*

'*Object?*' ('OBJECT...UO' in figure 4.27) of the first interpretation for 'NI' is a technical marker in the analysis grammar. This means that this case is not governed by the verb 'MOTOME-TA' and it should be governed by the other verb (in this case, the verb 'SHIMESU'). This is the same for 'KARA' in the second interpretation. Note that the verb 'MOTOME-TA' forms an embedded sentence and has a gap. Therefore, the noun 'IPPAN-KAI' is copied into the sentential structure marked with 'WO.'

The final step applies heuristic rules to compare several interpretations. In this example, the fist one is selected, because more NPs are governed by the verb 'MOTOME-

```
?  <?> <PARA>
|--?  <?>
|  |--S  <?>
|     |--ADVP  <hyou>  <NI>  <OBJECT...UO>
|     |--ADVP  <houtei-shiki>  <KARA>  <ORIGIN>
|     |--ADVP  <ippan-kai>  <WO>  <OBJECT>
|     |--VP  <motomeru>  <>  <GOV>
|        |--DSI  <motomeru>  <>  <GOV>
|        |--A  <ta>
|--?  <?>
   |--S  <?>
      |--ADVP  <hyou>  <NI>  <RECIPIENT>
      |--ADVP  <houtei-shiki>  <KARA>  <SUBJECT...UO>
      |--ADVP  <ippan-kai>  <WO>  <OBJECT>
      |--VP  <motomeru>  <>  <GOV>
         |--DSI  <motomeru>  <>  <GOV>
         |--A  <ta>
```

Figure 4.27: Example of PARA Special Node

TA' than the second one. The first one is two NPs ('KARA' and 'WO'), and the second one is one NP ('WO').

Tense and Aspect Analysis

Tense and aspect analysis phase consists of rewriting rules which use the expression, *if-then-else* form in substructure operation part. The rule in figure 4.28 is one of the rewriting rules used for the tense analysis. This rule determines the *deep* tense in the embedded sentence. Since a *relative* tense system is used in Japanese, tense in an embedded sentence is affected by the tense of the upper level sentence [Kusanagi82]. Therefore, this rule checks the upper level tense with

'S.J_UPLEVEL_TENSE = 'PAST''

in the matching condition part. Because we do not need to modify the tree structure in this phase, this rule only adds properties to the nodes. This is expressed with '**' in the creation part.

```
TENSE_SOTAI_PAST.rr;
 mi; level(0,0); top_to_bottom; left_to_right; order(1); tree;
 mc; %(( S # ));
      S.J_UPLEVEL_TENSE = 'PAST';
 cr;
  if S.J_DEEP_ASPECT = 'kanryou' | 'kekka';
   then   **;     S.J_DEEP_TENSE <= 'PPERFECT';
   else   **;     S.J_DEEP_TENSE <= 'PAST';
  end_if;
end_rr.TENSE_SOTAI_PAST;
```

Figure 4.28: Example of Tense Analysis Rule

Generating Dependency Structure

The former phases (11) generate a 'phrase structure' tree (figure 4.29). Phase 12 transforms the phrase structure tree into a 'dependency structure' tree (figure 4.30). Rules for this phase are easily expressed by tree-to-tree transformation rules of GRADE.

Result of Analysis Phase

The tree structure in figure 4.30 is the final result of the analysis phase. This tree structure expresses the *meaning* of the sample sentence as follows:

- The main predicate is 'ZOUKASURU (to increase).'
- It governs the noun 'SHINPO (advance)' by 'CAUSE' deep case.
- It also governs the noun 'JIDOU-KA-SEN (automated ship)' by 'SUBJECT' deep case.
- It is marked by 'TA (past).'
- It may govern 'SOURCE' and 'GOAL' deep cases. However, there is no such cases in the input sentence.
- The noun 'SHINPO' is modified by three nouns:
  1. 'DENKI-KEISOKU-HOU (electrical measurement)'
  2. 'DEETA-SHORI (data processing)'

```
? <?>
|--S <?> <> <MAIN>
|  |--ADVP <shinpo> <DE> <CAUSE>
|  |  |--NP <shinpo> <> <GOV>
|  |  |  |--ADJP <?> <> <SUBJECT>
|  |  |  |  |--NP <jidou-ka-kiki> <> <GOV>
|  |  |  |  |  |--N <denki-keisoku-hou>
|  |  |  |  |  |--PSYM <?>
|  |  |  |  |  |  |--PUNC <TOU_TEN>
|  |  |  |  |  |--N <deeta-shori>
|  |  |  |  |  |--PSYM <?>
|  |  |  |  |  |  |--PUNC <TOU_TEN>
|  |  |  |  |  |--N <jidou-ka-kiki> <> <GOV>
|  |  |  |  |--BKK <no> <NO> <SUBJECT>
|  |  |  |--N <shinpo> <> <GOV>
|  |  |--BKK <de> <DE> <SUBJECT...UO>
|  |--ADVP <jidou-ka-sen> <GA> <SUBJECT>
|  |  |--N <jidou-ka-sen> <> <GOV>
|  |  |--BKK <ga> <GA> <SUBJECT...UO>
|  |--VP <zoukasuru> <> <GOV>
|  |  |--DSI <ZOUKASURU> <> <GOV>
|  |  |--A <ta>
|  |--ADVP <?> <KARA> <SOURCE>
|  |  |--N <?> <> <GOV>
|  |--ADVP <?> <NI...UO> <GOAL>
|     |--N <?> <> <GOV>
|--PUNC <KU_TEN>
```

Figure 4.29: Phrase Structure Tree

```
? <?>
|--DSI <zoukasuru> <> <MAIN>
   |--N <shinpo> <DE> <CAUSE>
   |  |--N <?> <no> <SUBJECT>
   |     |--N <denki-keisoku-hou> <> <PARAELEMENT>
   |     |--N <deeta-shori> <> <PARAELEMENT>
   |     |--N <jidou-ka-kiki> <> <PARAELEMENT>
   |--N <jidou-ka-sen> <GA> <SUBJECT>
   |--A <ta>
   |--N <?> <KARA> <SOURCE>
   |--N <?> <NI...UO> <GOAL>
```

Figure 4.30: Dependency Structure Tree

3. 'JIDOUKA-KIKI (automated equipment).'

- The semantic relation between 'SHINPO' and three nouns is 'SUBJECT.'

4.3.2 Japanese-into-English Transfer

The Japanese into English Transfer phase of the Mu project [Nagao85c] has five components:

1. Simple word insertion (for noun).
2. Pre-transfer loop.
3. Predicate transfer by dictionary rules.
4. Word selection by co-occurred words.
5. Post-transfer loop.

Complex examples in the transfer phase are discussed in section 4.2.4. In this section, outline of this phase is explained.

Simple Word Insertion

The transfer system first looks up the bilingual dictionary for nouns (include compound nouns) and inserts the translated words. In abstract of technical papers, many nouns are technical terms and most of them have direct translations. Therefore, this is done without any word selections by 'dictionary look-up' sub-system of GRADE system, not by the GRADE grammar rules.

Figure 4.31 shows the result of dictionary look-up, that is, the input tree structure of the sample sentence to the transfer grammar, whose nodes for nouns already have translations. The first and second fields are Japanese part-of-speech and lexical unit. The third and fourth fields are English part-of-speech and lexical unit.

Note that the words selected in this phase are default translations. If other rules for the word selection are described in the grammar or the dictionary, those rules may change translations as discussed in section 4.2.4.

```
J-DSI  <zoukasuru>
|--J-N  <shinpo>  <N>  <advance>
|  |--J-N  <?>
|     |--J-N  <denki-keisoku-hou>  <N>  <electrical measurement>
|     |--J-N  <deeta-shori>  <N>  <data processing>
|     |--J-N  <jidou-ka-kiki>  <N>  <automatic equipment>
|--J-N  <jidou-ka-sen>  <N>  <automated ship>
|--J-A  <ta>
|--J-N  <?>
|--J-N  <?>
```

Figure 4.31: Input Tree Structure for Transfer Grammar

Pre-Transfer Loop

In this phase, the rules for the adjustment of syntactic structures are applied to generate a *natural* translation (see section 4.2.4). This adjustment is based on Japanese syntactic/semantic heuristics.

Predicate Transfer by Dictionary Rules

A predicate, such as a verb or a (predicative) adjective, is transferred by the tree-to-tree rewriting rules stored in the transfer dictionary, because it needs a word selection and a complex mapping from Japanese surface/deep cases into English surface/deep cases. The dictionary rules select not only the English deep cases but the English surface cases (*subject, object* and prepositions). The surface cases chosen in this phase are default cases for the English generation. The grammar for the generation may change them.

Figure 4.32 shows the result of the dictionary rule application for the verb 'ZOUKA-SURU.' In general, dictionary rules are generated from a formatted dictionary. Details to generate 'standard' dictionary rules from the formatted dictionary are discussed in chapter 5. In this case, the English verb 'to increase' is selected as the translation for the Japanese verb 'ZOUKA-SURU.' The deep/surface cases are mapped as:

CAUSE ⇒ *deep*: CAU, *surface*: by
SUBJECT ⇒ *deep*: OBJ, *surface*: SUBJ

```
J-DSI  <zoukasuru>  <V>  <increase>
|--J-N  <shinpo>  <N>  <advance> <CAU> <by>
...
|--J-N  <jidou-ka-sen>  <N>  <automated ship> <OBJ> <SUBJ>
|--J-A  <ta>
```

Figure 4.32: Result of Predicate Transfer by Dictionary Rule

```
?  <?>
|--V  <increase> <PAST> <SUPERT>
   |--N  <advance> <CAU> <by>
   |  |--N  <> <AGT> <in>
   |     |--N  <electrical measurement>
   |     |--N  <data processing>
   |     |--N  <automatic equipment>
   |--N  <automated ship> <OBJ> <SUBJ>
```

Figure 4.33: Result of Transfer Phase

Word Selection by Co-occurred Words

The core part of this word selection is stored as 'dictionary rules.' However, the timing of invocation of dictionary rules is determined by the transfer grammar, which refers to the attributes of each word added by the dictionary look-up sub-system. Details are discussed in chapter 5.

Post-Transfer Loop

In this phase, the rules for the adjustment of syntactic structures are applied to generate a *natural* translation (see section 4.2.4). This adjustment is based on English syntactic/semantic heuristics in contrast to the 'pre-transfer loop.'

Result of Transfer Phase

After the application of the post-transfer loop rules, the final result of the transfer phase is obtained. Figure 4.33 shows the result for the sample sentence. This tree shows that:

- The main verb is 'to increase.' Its tense and aspect are 'PAST' and 'SUPERT.'

- 'To increase' governs two noun phrases: 'advance' and 'automated ship.'
- The deep case of 'advance' is 'CAUse' and its default preposition (surface case) is 'by.'
- The deep case of 'automated ship' is 'OBJect' and its default surface case is 'SUBJect.'
- 'To increase' is modified by three noun phrases: 'electrical measurement', 'data processing' and 'automatic equipment.' Semantic relation between them is 'AGT (agent)' and its default preposition is 'in.'

Note that several English surface structures are already determined by this transfer phase (e.g. the preposition 'by'). However, these are tentative and the English generation phase may modify them.

4.3.3 English Generation

In the English Generation phase of the Mu project, five tasks are executed:

1. Dictionary look up for English Generation.
2. Generation of sentential structures (mapping from deep cases into surface cases).
3. Word specific transformations.
4. Heuristic determination of the number and the articles of the noun phrases.
5. Morphological generation. This phase is implemented in LISP directly, in stead of using GRADE grammar rules.

Rules for each task are expressed with subgrammar networks and subgrammars. Note that the generation is not sequentially executed by this order, because one rule calls other rules recursively. For example, the rules for generation of sentential structures (2) calls the rules for word specific transformations (3), if necessary.

```
V  <increase>
|--N  <advance>  <SUBJ>  <CPO>
|  |--N  <?>  <SUBJ>  <AGT>
|     |--N  <electrical measurement>
|     |  |--?  <?>
|     |     |--ADJ  <electrical>
|     |     |--N  <measurement>
|     |--N  <data processing>
|     |  |--?  <?>
|     |     |--N  <data>
|     |     |--N  <processing>
|     |--N  <automatic equipment>
|        |--?  <?>
|           |--ADJ  <automatic>
|           |--N  <equipment>
|--N  <automated ship>  <OBJ1>  <OBJ>
   |--?  <?>
      |--V  <automate>
      |--N  <ship>
```

Figure 4.34: Result of Dictionary Look-up in English Generation Phase

Dictionary Look up for English Generation

The first task in the English Generation phase is looking up the English dictionary by the dictionary sub-system. The result of the dictionary look up is shown in figure 4.34. In the transfer phase, a compound word is treated as a single entry. However, the dictionary look up in the generation phase adds the detailed structure of a compound word, because the generation phase needs this information. For example, the tree structure in figure 4.34 shows that

> the word **'electrical measurement'** is a compound word whose components are the ADJective **'electrical'** and the Noun **'measurement.'**

Generation of Sentential Structures

The generation of sentential structures is not straight-forward, because many obligatory case elements in English are omitted in Japanese. Therefore, the generation of sentential structures is divided into 2 steps:

```
increase_V_GENER_2.rr;
 mi; level(0,0); order(1); tree;
 mc; %((V CASE));
   CASE: %(X1 X2); X1,X2:disorder_skip( %( ? ) );
   X1.E_DEEP_CASE = 'CPO'; X2.E_DEEP_CASE = 'OBJ';
   V.E_UID = '2';
  cr; **;
   X1.E_SURFACE_CASE <= 'SUBJ'; X2.E_SURFACE_CASE <= 'OBJ1';
   X2.E_NOUN_ATTACH <= 'NUMBER';
   V.E_MAIN_VERB <= 'increase'; V.E_VERB_TYPE <= 'VERBONLY';
end_rr.increase_V_GENER_2;
```

Figure 4.35: Dictionary Rule for the Verb 'to increase'

1. Mapping from deep cases into surface cases depending on a specific verb. This rule is expressed as a dictionary rule.

2. Applying general rules to adjust the structure. For example, passive voice is chosen for the generation, if there is a *deep* object but no *deep* case (e.g. agent) which can be generated as a *surface* subject.

The first step tries to use the deep cases as much as possible. The rewriting rule shown in figure 4.35 is the dictionary rule for the verb **'to increase.'** This rule maps the deep cases into surface cases as follows:[12]

CPO (Causal Potency) ⇒ SUBJect,

OBJect ⇒ OBJ1 (direct object)

Note that this rule put the mark 'NUMBER' to the direct object in the creation part:

'X2.E_NOUN_ATTACH <= 'NUMBER'.

This mark is used in the word specific generation phase, which is called from the rules in the sentential generation phase.

Figure 4.36 shows the final result of the determination of sentential structures. The result in this figure is already applied to the rules for the word specific transformation, which is explained in the next.

[12]This rule 'increase_V_GENER_*2*' is applied to the structure which does not have an AGT (agent) deep case. When the structure has an agent, the rule 'increase_V_GENER_*1* maps the agent to a surface subject.

```
S  <?>
|--NP  <?>  <SUBJ>  <CPO>
|  |--N  <advance>  <SUBJ>  <CPO>
|     ...
|--V  <increase>
|--NP  <?>  <OBJ1>  <OBJ>
   |--N  <number>  <OBJ1>  <OBJ>
      ...
```

Figure 4.36: Determination of Sentential Structure

```
NUMBER_ATTACH.rr;
 mi; level(0,0); order(1); tree;
 mc; %((X1 #));
   X1.E_CAT = 'N'; X1.E_NOUN_ATTACH = 'NUMBER';
   not(X1.E_SEM = 'MO'|'MN'|'MU'|'MS');
   X1.E_NUMBER_TYPE = 'C'|'GC';
 cr; %((N (X1 #)));
   N.* <= X1.*; N.E_LEX <= 'number'; N.E_NUMBER_TYPE <= 'U';
   X1.E_SURFACE_CASE <= 'A_COMPO'; X1.E_PREP_LEX <= 'of';
   X1.E_DEEP_CASE <= 'VL1'; X1.E_NUMBER <= 'PLURAL';
end_rr.NUMBER_ATTACH;
```

Figure 4.37: Rule to Insert a Phrase 'number of'

Word Specific Transformation

To generate a *natural* sentence, we need a lot of heuristic rules that depend on specific words. The rule shown in figure 4.37 is one of such word specific rewriting rule.[13]

This rule first checks the mark

```
E_NOUN_ATTACH = 'NUMBER'.
```

If found, it checks the semantic marker and the number type of the noun. When the semantic marker of the noun is not related to the 'NUMBER', which is expressed as semantic markers `MO`, `MN`, `MU`, and `MS`, and the noun is 'Countable', this rule inserts the

[13]Strictly speaking, the rule in figure 4.37 is not directly depend on a specific word. This rule is applied to the tree, when the noun is marked with 'E_NOUN_ATTACH = 'NUMBER'.' This mark is put by the rules for specific verbs (e.g. 'to increase' and 'to decrease') as shown in figure 4.35. Therefore, this is also one of the word specific transformation rules.

```
N  <number>  <OBJ1>  <OBJ>
|--N  <number>  <OBJ1>  <OBJ>
|--PP  <?>  <A_COMP0>  <VL1>
   |--PREP  <of>
   |--NP  <?>  <A_COMP0>  <VL1>
      |--N  <automated ship>  <A_COMP0>  <VL1>
```

Figure 4.38: Result of Heuristic Insertion of a Phrase 'number of'

expression '**number of**' to the noun and put 'PLURAL' mark to the noun. This shows the heuristic rule:

> The object of the verbs '**to increase**' and '**to decrease**' must be the noun related to the concept of '*number.*' If the object is not such a word, the phrase '**number of**' should be inserted.

In the sample sentence, the object of the verb '**to increase**' is the noun '**automated ship**', which is not directly related to the concept of '*number.*' Therefore, this rule transforms the phrase as:

> **to increase an/the automated ship. ⇒ to increase the number of automated ships.**

Figure 4.38 shows the result of this transformation.

Heuristic Determination of Number and Article

In Japanese, there is no simple expressions corresponding to English number and articles. Therefore, the translation system must decide them during the translation process. To determine them, we can consider two methods:

1. Using semantics, contextual information and inference rules to find the number of nouns and definiteness of them.

2. Using heuristic rules which mainly refer to the syntactic information.

The first one is an ideal solution. However, it is not so easy to apply it to a *practical* system. Therefore, the English generation grammar of the Mu project uses heuristic rules to determine the number of nouns and the article.

```
DEFAULT_ART.sg;
   sg_mode; order(2); deterministic;
   rr_in_sg;
     A1_ART; CARD_ART; ORD_ART; CCC_ART; EE_ART;
      D_ART; C_ART; B_ART; BB_ART;
   DEFT_ART; SS_ART; THERE_ART;
end_sg.DEFAULT_ART;
```

Figure 4.39: Subgrammar to Determine Articles

```
D_ART.rr;
  mi; level(0,100);order(2,skip); tree;
  mc; %((NP1 (N1 (DIC ADJ N2) ) (PP PREP NP2) #1));
      PREP.E_LEX = 'of';
      N1.E_NUMBER = 'SINGULAR'; N1.E_NUMBER_TYPE = 'C';
      DIC.$SYS$ = 'DICCOMP';
  cr; %((NP1 ART (N1 (DIC ADJ N2) ) (PP PREP NP2) #1));
      ART.E_LEX <= 'a'; ART.E_SUBCAT <= 'INDEFINITE';
end_rr.D_ART;
```

Figure 4.40: Rewriting Rule to Determine Articles

Figure 4.39 shows the subgrammar to determine articles of noun phrases by heuristics. This subgrammar contains several rewriting rules and they are ordered according to their precedence. One of the heuristic rules contained by this subgrammar is shown in figure 4.40.

The rewriting rule D_ART in figure 4.40 finds a compound noun, whose components are an adjective and a countable noun, followed by a prepositional phrase whose preposition is 'of.' When such a noun is found, this rule uses 'a' for the article.[14]

Result of Syntactic Generation Phase

After the application of the rules for heuristic determination of the number and the articles, the tree structure shown in figure 4.41 is obtained. This tree structure is also the final result of the English generation phase. It, for example, shows:

- 'advance' is plural without the article (ART <?>).

[14]Final choice between 'a' and 'an' is determined at the morphological generation phase.

```
S  <?>
|--NP  <?>  <SUBJ>  <CPO>
|  |--ART  <?>
|  |--N  <advance>  <SUBJ>  <CPO> <PLURAL>
|  |--PP  <?>  <SUBJ>  <AGT>
|     |--PREP  <in>
|     |--NP  <?>  <SUBJ>  <AGT>
|        |--NP  <?>
|        |  |--N  <electrical measurement> <SINGULAR>
|        |--CONJ  <,>
|        |--NP  <?>
|        |  |--N  <data processing> <SINGULAR>
|        |--CONJ  <and>
|        |--NP  <?>
|           |--N  <automatic equipment> <SINGULAR>
|--V  <increase> <PAST> <PLURAL> <THIRD>
|--NP  <?>  <OBJ1>  <OBJ>
   |--ART  <the>
   |--N  <number>  <OBJ1>  <OBJ> <SINGLUAR>
   |--PP  <?>  <A_COMPO>  <VL1>
      |--PREP  <of>
      |--NP  <?>  <A_COMPO>  <VL1>
         |--ART  <?>
         |--N  <automated ship>  <A_COMPO>  <VL1> <PLURAL>
```

Figure 4.41: Result of Determination of Number and Articles

```
================================================================
E82060001_3_1 (05/07/87, 05/07/87, 05/07/87)
----------------------------------------------------------------
電気計測法、データ処理、自動化機器の進歩で自動化船が増加した.
----------------------------------------------------------------
 Advances in electrical measurement, data processing and automatic
equipment increased the number of automated ships.
----------------------------------------------------------------
```

Figure 4.42: Input Sentence and the Result of Translation

- **'electrical measurement'** is singular without the article.
- **'to increase'** is past, plural and third-person.
- **'number'** is singular with the definite article **'the.'**

Morphological Generation

Finally the English morphological generation program, which is implemented in LISP directly, generates the result of the translation. Figure 4.42 shows the input sentence and the result of translation.

4.4 Summary of this Chapter

In this chapter, we discussed typical problems in MT grammars and their solutions with the GRADE system to demonstrate the usefulness of GRADE. The problems we discussed are:

- In the analysis of English sentences:
  1. Ambiguity of parts-of-speech.
  2. Complex constructions of sentences.
  3. Structural ambiguities.
- In the transfer from Japanese into English:
  1. Word selection.

2. Adjustments of syntactic structures

The ways to solve these problems are shown and they are based on the features of the GRADE system:

1. Tree-to-tree transformation with flexible pattern matching function.

2. Procedural grammar approach with subgrammar and subgrammar network.

3. Dictionary rules.

We also discussed the grammars (from Japanese into English) developed in the Mu project to show how these features of GRADE are used in a practical grammar. This shows that the design objectives of the GRADE system discussed in section 3.2 are appropriate to develop a practical machine translation system.

102 項欠

Chapter 5

Utilization Method of Dictionary Databases

5.1 Dictionary Database

A dictionary database is a key component of a machine translation system. Many grammatical rules need to refer to the lexical information stored in the dictionary to decide their proper action. Also, for any practical machine translation system, the size of this dictionary database must be very large; for example, the number of lexical entries, is more than 80,000 in the Mu project. Thus, techniques for developing and utilizing the dictionary become more important than in a small experimental systems.

Furthermore, from the view point of the research and development of a natural language processing system, a large dictionary database is a significant resource. If a database is developed *independent* of any specific system, we can then share that database with many natural language processing systems. This is also important for future research and development [Yoshida86].

In this chapter, we will first discuss the properties of dictionary databases that are necessary for natural language processing. The configuration of a *neutral* and *general purpose* dictionary database developed under the Mu project [Nagao83b] that is based on those properties is then described. A software system for transforming the *neutral* dictionary into *specific* dictionaries for Japanese-English and English-Japanese machine translation systems in the Mu project is also presented.

5.2 Dictionaries for Natural Language Processing

When we develop a dictionary database for natural language processing, the following two methods can be used:

1. Taking a computer readable *ordinary* dictionary compiled for humans [Nagao80b], like LDOCE (Longman Dictionary of Contemporary English [Longman78]), and transforming it into a form suitable for the processing required.

2. Developing a *special* dictionary for a *specific* system which is under development. This is the usual way of making a dictionary for a natural language processing system.

The first method of using an *ordinary* dictionary offers the possibility of obtaining quite varied information for a variety of purposes, because an ordinary dictionary is not based on any specific computational model. However, since it is, of course, compiled for human usage, the format of its description is generally too free for it to be used by a computer directly. It takes a good deal effort to transform such a description to an adequate form for natural language processing.[1]

On the other hand, a computer dictionary designed for a *specific* system is optimal only for that system. In addition, it is more dependent on the particular grammar of that system and has only the information necessary for that system. Even though a dictionary database may sometimes involve procedural descriptions, (that is, a kind of program, which provide much flexibility [Isahara84]) since such programs depend deeply on the algorithm of the particular system, it becomes impossible to use such a dictionary in different natural language processing systems.

Both these methods, then, have merits and demerits. Therefore, to realize an *intermediate* dictionary database, 'a *neutral* and *general purpose* dictionary', we have designed a dictionary based on the following objectives:

[1] [Tsurumaru84a] and [Tsurumaru86] use a Japanese dictionary to generate a thesaurus of nouns. [Nagao82c] uses LDOCE to parsing English sentences. [Alshawi87] and [Boguraev87] uses LDOCE to extract dictionary items for GPSG [Gazdar85]. [Nakamura86a], [Nakamura86b], [Nakamura87b], [Nakamura87c], and [Nakamura88] also use LDOCE to generate a thesaurus of verbs and nouns.

1. The dictionary database should be developed and maintained *independently* from the translation system. Also, it should be usable for various purposes. When developing such a general purpose dictionary, each word is described in terms of its linguistic information only. Then, to use the dictionary database, this description is transformed into a form specific to the translation process. This not only makes the description clear and understandable for humans, but also, and more importantly, it allows us to use the same dictionary database for different purposes, for example, in sentence analysis and sentence generation.

2. A word's description may contain *structured* information. This makes it possible to express complex linguistic information in the dictionary. However, it should also be easy for word descriptions to be handled by a program. Even if new content is added to a description, the modification should only affect words which should have the new content. Unwanted effects of the modification must not spread through the whole dictionary.

3. The linguistic phenomena *specific* to a word should be expressed in detail. Such phenomena are expressed descriptively and are later transformed into rewriting rules for the purposes of the translation grammar.

5.3 Dictionaries in Mu Project

In the Mu project, four dictionary databases (Japanese, Japanese-to-English, English-to-Japanese, and English) have been developed. Each item of these dictionaries is first written in a lexicon entry form by a lexicographer. As shown in figure 5.1, a lexical description of a word written in the entry form is converted into a computer readable form for input to the system. Then the data is inserted into the dictionary database. Finally, the description is transformed into the specific forms used by the translation grammars of the Mu project [ETL84], which are explained in chapter 4.

The English *general purpose* dictionary database, for example, is transformed into a processing dictionary for the English *analysis* phase. It is also transformed into a dictionary for English *generation* (figure 5.2). This shows that a general purpose

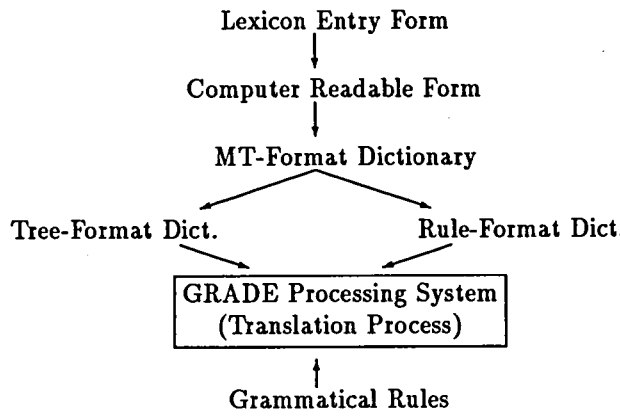


Figure 5.1: Flow of Data Transformations of Dictionaries

dictionary can be transformed into dictionaries which are used in completely different ways.

In the next section, the content of the general purpose dictionary (which is called the 'MT-format dictionary') and the processing dictionaries for the Mu project are described. A software system for dictionary transformation is also discussed.

5.4 General Purpose Dictionary Database

The *neutral* and *general purpose* dictionary database is first developed by lexicographers, who fill the columns of a lexicon entry form such as that shown in figure 5.3. The content of this form consists of purely linguistic information: part-of-speech, inflection type, derivations, verb pattern[2], case-frame information, etc. It is *independent* of specific processing systems. Therefore, the lexicographers only need to take account of the purely linguistic features of words. They do not need to know the computational details of the processing system. This increases the efficiency of dictionary development.

The lexical information written in the entry form is then entered into the system and converted to the computer readable format shown in figure 5.4 and 5.5.[3] We will

[2]Verb pattern of the Mu project is based on the grammartical code used in LDOCE [Longman78] [Nagao82c]: 'I' for intransitive verbs, 'T' for transitive verbs, etc.

[3]In figures of this chapter, Romaji is used for expressing Japanese words. In the real system, however, Japanese words are expressed with Kana and Kanji combinations.

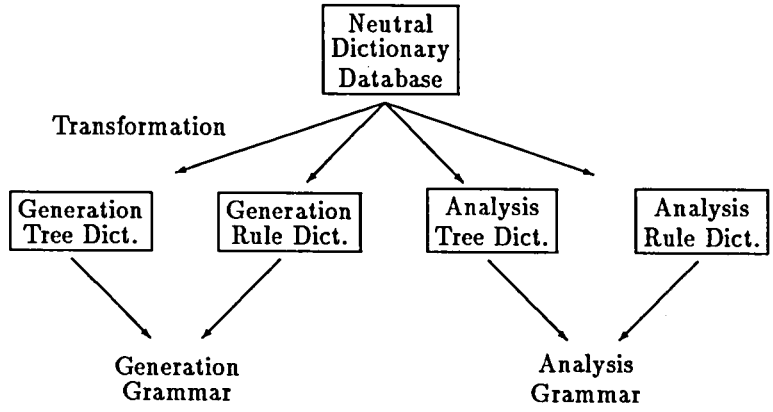


Figure 5.2: Transformation of Dictionary Database into Processing-specific Dictionaries

call this format the MT-format (MTF). As can be seen from figure 5.3 and 5.4, the content of the MT-format is equivalent to that of the entry form. The MT-format dictionary is directly managed by the dictionary software system. In this section, the content of this MTF dictionary is only described sufficiently for the discussion which follows; the detailed linguistic information of the entry form and the instructions for the lexicographers appear in [JICST84] [Nagao85c].

Figure 5.4 shows the content of the English MTF dictionary for the verb 'affect'. In this format, 3 principal pieces of linguistic information concerning English verbs are expressed:

(a) morphological information:

INFLECTION type of this verb is REGULAR

(b) information about DERIVATION:

verbal derivation (STATE) of this verb is 'effect'

(c) information about case frame (C-FRAME):

surface subject (SUBJ)

$\Longleftrightarrow$ deep causal potency (CPO)

No.	16				
Lexical Unit	affect				
Sub-Category	VERB				
Component	Main Verb	affect			
	Adverbial Particle				
	Adverb-Correlative				
	Preposition				
	Prep-Correlative				
Subject Code					
Usage ID	1				
Semantic Category					
Verb Pattern	I / L / (T) / V / X / D				
Aspect	STATive / TRANsitive / (PROCess) / COMPLeTive / MOMentary				
Voice	++ / (+) / -	Subject in D-verb passive	1 / 2		
Volition	+ / (-)				
Agent of to-inf.	SUBJect / OBJect				

	Surface Case	Deep Case	Syntactic Form	Semantic Codes	
1	SUBJ	CPO			1
2	OBJ1	OBJ			1
3					
4					

Remarks

Figure 5.3: Example of Lexicon Entry Form

```
1  ((SEQ 16)
2   (E_LEX affect)                    ; English Lexical Unit
3   (E_CAT V)
4   (INFLECTION REGULAR)              ; (a) Inflection Type
5   (USAGE
6     ((E_UID 1)
7      (DERIVATION                    ; (b) Info. about derivations
8         (E_DERIV_STATE_ACTION effect)
9         (E_DERIV_STATE_ACTION_UID 1))
10     (COMPONENT (E_MAIN_VERB affect))
11     (E_VERB_PAT T)
12     (E_VERB_ASPECT PROC)
13     (E_VERB_VOICE /+)
14     (E_VERB_VOLITION /-)
15     (C-FRAME                       ; (c) Info. about Case Frame
16        ((E_SURFACE_CASE1 SUBJ)     ; Surface Case
17         (E_DEEP_CASE1 CPO)         ; Deep Case
18         (E_OBLIGATORY1 1))
19        ((E_SURFACE_CASE2 OBJ1)     ; Surface Case
20         (E_DEEP_CASE1 OBJ)         ; Deep Case
21         (E_OBLIGATORY1 1))))))
```

Figure 5.4: MT Format Dictionary of the English Verb "affect"

```
1  ((SEQ 6238)
2   (J_LEX eikyou)                    ; Japanese Lexical Unit
3   (J_CAT meishi)
4   (USAGE
5    ((E_LEX effect)                  ; (d) Translation Equivalent
6     (E_UID 1)                       ;     if no condition
7     (E_CAT N)

8     (MODIFIED                       ; (e) Word Selection by
                                      ;     a Modified Phrase
9       ((J_BKK_LEX he)               ;   N1 HENO EIKYOU
10       (J_DEEP_CASE ukete)          ;       -> effect on
11       (E_PREP_LEX on)
12       (E_LEX effect)
13       (E_CAT N)))

14    (CORRESPONDENCE                 ; (f) Word Selection by
                                      ;     a Governing verb
15      ((J_LEX ataeru)               ; N1 GA N2 NI EIKYOU WO ATAERU
16       (E_LEX affect)               ;     -> N1 affect N2
17       (E_CAT V)
18       (E_UID 1)

19       (CASE_FRAME                  ; (g) Case Frame Mapping
20          ((J_SURFACE_CASE ga)      ; between "ATAERU" and "affect"
21           (J_DEEP_CASE shutai)
22           (E_SURFACE_CASE SUBJ)
23           (E_DEEP_CASE CPO))
24          ((J_SURFACE_CASE wo)
25           (J_DEEP_CASE taishou)
26           (J_POS_FLAG DEL))       ; "EIKYOU" is an OBJECT
27          ((J_SURFACE_CASE ni he)
28           (J_DEEP_CASE ukete)
29           (E_SURFACE_CASE OBJ1)
30           (E_DEEP_CASE OBJ))))
31      ))))
```

Figure 5.5: MT Format Dictionary of the Japanese Noun "EIKYOU" from Japanese into English

surface direct object (OBJ1)

$\Longleftrightarrow$ deep object (OBJ)

Figure 5.5 shows the content of the Japanese-into-English transfer MTF dictionary for the noun 'EIKYOU (effect)'. In this case, the following 3 correspondences between Japanese and English are expressed:

(d) translation equivalent, in the case of there being no special condition (E_LEX).

EIKYOU $\longrightarrow$ effect

(e) word selection by a modified phrase (MODIFIED).

N HENO EIKYOU $\longrightarrow$ effect on N

(f) word selection by a verb (CORRESPONDENCE). Note that the case frame mapping is expressed in the (g) CASE_FRAME part.

N1 GA N2 NI EIKYOU WO ATAERU $\longrightarrow$ N1 affect N2

The content of the MTF dictionary is represented by a list of property names (e.g. E_CAT, stands for 'English Category') and property value (e.g. N, stands for 'Noun'), which shows types of information and appropriate values. This form of expression allows the system to extract dictionary information and to insert new information easily. Also, a property value can be a list of property names and values and so on recursively, as in an ordinary frame-like representation. Therefore, structural information such as

a verb has several case frames and each case frame has an internal structure which is a mapping between surface case and deep case and its constraints.

can be naturally expressed.

5.5 Dictionaries for Translation Processes

The MTF dictionary is *independent* of the dictionaries for the translation process. The dictionaries which are accessed by the translation grammar are transformed automatically from the MTF dictionary, as shown in figure 5.1. This transformation phase

provides both for the independence of a general purpose dictionary database maintained by lexicographers, and for flexible and effective use of dictionaries for specific translation systems.

A processing dictionary system consists of

1. Tree-format dictionary and

2. Rule-format dictionary.

The *tree-format* dictionary contains the passive information, which is referred to by the grammatical rules. The *rule-format* dictionary is expressed by tree-to-tree rewriting rules which handle word specific phenomena. These dictionary rules are applied during the translation process.

5.5.1 Tree-format Dictionary

The *Tree-format* dictionary contains linguistic information which is referred to by general grammatical rules, such as part-of-speech and derivations. This information is extracted from the MTF dictionary and is arranged for the grammar of the translation system. The information is classified as follows:

1. information explicitly written in the MTF dictionary

2. information not explicitly written in the MTF dictionary, but which can be generated from it.

The tree-format dictionary stores these as a tree structure which is handled by the grammar writing system GRADE [Nakamura84c] [Nakamura85b].

As an example of this tree-format dictionary, the tree structure for the noun 'EIKYOU' in the Japanese-to-English processing dictionary is shown in figure 5.6.[4] This tree structure has the following information, corresponding to the content of the MTF dictionary in figure 5.5:

[4]The structure shown in figure 5.6 seems not to be a *tree*, but a list of property-name and value pairs. However, it is handled as a *tree* whose top node contains the list of property-name and value pairs. This is useful when we need to express complex and structural information.

```
1  (((E_CAT (N))
2    (E_LEX (effect)                  ; (a) Standard Translation
3    (E_UID (1))
4    (J_DIC_VARIANT (T))              ; (b) Flag for a word selection
                                      ; by a modified phrase
5    (J_DIC_DERIV (ataeru))           ; (c) Flag for a word selection
                                      ; by a governing verb
6    ))
```

Figure 5.6: Tree-format Dictionary for the Japanese Noun "EIKYOU" from Japanese into English

(a) standard translation, if this has no special condition for word selection (E_CAT, E_LEX, E_UID). The information of (d) in figure 5.5 is transformed into lines 1–3 in figure 5.6.

(b) a flag which expresses the existence of information about a modification. This flag has a value of J_DIC_VARIANT. It is T (line 4 in figure 5.6). when the MTF dictionary contains a MODIFIED part as in (e) in figure 5.5. If this word is modified by a specific phrase in an input sentence, it will be treated specially. The exact way to treat it is expressed in the rule-format dictionary as discussed later. This flag is not expressed in the MTF dictionary explicitly.

(c) a flag which expresses the existence of information about a verb co-occurrence (J_DIC_DERIV). When the MTF dictionary contains a CORRESPONDENCE part as in (f) in figure 5.5, the value of this flag is the head word of a co-occurring verb (line 5 in figure 5.6). In this example, if this noun 'EIKYOU' is governed by a verb 'ATAERU' in an input sentence, it is treated specially as for flag (b). This flag is also not expressed in the MTF dictionary explicitly.

Detailed processing for cases (b) and (c) is stored as grammar rules in the rule-format dictionary for the Japanese-to-English transfer dictionary. The flags J_DIC_VARIANT and J_DIC_DERIV tell the grammar whether this word has word specific rules or not. The grammar, which treats general linguistic phenomena, checks these flags. Then, if necessary, it calls the grammatical rules directly to process word specific phenomena.

These flags do not express linguistic features of a word. They are instead required by the *specific* translation system, that is, the translation grammar of the Mu project. Therefore, these flags are not written in the MTF dictionary, but are generated when the tree-format dictionary is transformed from the MTF dictionary.

Note that the example in this section is very simple. Although it may appear to be a list of information rather than a tree, if the MTF dictionary contains more complex information than this example, it is actually a tree. The Japanese MTF dictionary has information about compound words. In such a case, each stem of a compound word is a sub-tree in the Tree-format dictionary.

5.5.2 Rule-format Dictionary

As mentioned in section 5.4, the MTF dictionary database contains the following information:

1. case frame correspondence between Japanese and English verbs.

2. case frame of Japanese and English verbs (correspondence between surface cases and deep cases).

3. co-occurrence between words (e.g. a noun and a verb).

This kind of complex and word specific information is declaratively expressed in the MTF dictionary. It is not a good idea to store this complex and structured information directly in the tree-format dictionary and to interpret it by *general* rules for translation. It is more effective to transform this information into *word specific* grammatical rules (tree-to-tree transformation rule) beforehand, and to call this transformed rule during the translation process if necessary. Therefore, the rule-format dictionary in the Mu project contains such complex linguistic information in the form of rewriting rules of the GRADE system [Nakamura83] [Nakamura85b].

Information of each word stored in the MTF dictionary is expressed declaratively and independent of the way it is used by the system. One and the same description can be transformed into specific rewriting rules for various purposes. The description

```
1  affect_VG_ANALYSIS_V_1.rr; /* Rewriting Rule for English Analysis  */
2   mi;  level(0,0); order(1); tree;
3   mc;  %((SENT CASES));
4        SENT.E_UID = '1';
                      -
5        CASES: %(X1 VG X2); X1,X2:disorder_skip( %( X ) );
6        X1.E_SURFACE_CASE = 'SUBJ'; /* Constraint for surface case */
                             ----
7        X2.E_SURFACE_CASE = 'OBJ1';
                             ----
8   cr; **;
9         X1.E_DEEP_CASE <= 'CPO';    /* Assignment of deep case */
                             ---
10        X2.E_DEEP_CASE <= 'OBJ';
                             ---
11        X1.E_GOVERNOR_ID <= VG.E_NODE_ID;
12        X2.E_GOVERNOR_ID <= VG.E_NODE_ID;
13        SENT.APPLIED <= 'T';
14  end_rr.affect_VG_ANALYSIS_V_1;
```

Figure 5.7: Dictionary Rule of English Analysis for the Verb "**affect**"

of a case frame for an English verb, for example, is used for the following 2 different purposes:

1. grammatical rule for English generation (mapping from deep-case into surface-case)

2. grammatical rule for English analysis (mapping from surface-case into deep-case)

Grammatical rules for analysis and generation are shown in figure 5.7 and 5.8. These are transformed from the sample MTF description of the English verb '**affect**' that is given in figure 5.4 as an example.

Figure 5.7 shows the rewriting rule of '**affect**' for English analysis. Lines 3–7 express the condition of this rule's application, and lines 8–13 specify the result. Thus, this rule transforms tree structures as shown in figure 5.9. Several properties may be added to the tree nodes during transformation. In this rule, for example, the underlined fields (`1`, `SUBJ`, `OBJ1`, `CPO`, and `OBJ`) are filled with the value of the MTF dictionary in

```
affect_V_GENER_1.RR; /* Rewriting Rule for English Generation */
 MI;  LEVEL(0,0); ORDER(1); TREE;
 MC;  %((V CASE));
      CASE: %(X1 X2); X1,X2:DISORDER_SKIP( %( ? ) );
      X1.E_DEEP_CASE = 'CPO';     /* constraint for deep case */
      X2.E_DEEP_CASE = 'OBJ';
      V.E_UID = '1';
 CR; **;
      X1.E_SURFACE_CASE <= 'SUBJ'; /* Assignment of surface case */
      X2.E_SURFACE_CASE <= 'OBJ1';
      V.E_MAIN_VERB <= 'affect';
      V.E_VERB_TYPE <= 'VERBONLY';
END_RR.affect_V_GENER_1;
```

Figure 5.8: Dictionary Rule of English Generation for the Verb "affect"

figure 5.4, while other parts are common to all rewriting rules in the English case frame analysis:

```
X1.E_GOVERNOR_ID <= VG.E_NODE_ID;
```

Figure 5.8 is the rewriting rule of 'affect' for English generation. This rule executes the inverse of the tree transformation of figure 5.9. This shows that the same description in the MTF dictionary may be used as a condition, and also as a result of the tree-to-tree transformation.

This example is a relatively simple dictionary rule compared with the rules really used in the Mu project. When the MTF dictionary contains more complex information (for example, co-occurrence of a specific preposition), the rewriting rule generated from the MTF dictionary becomes more complex.

5.6 Software Tools for Format Transformation

As mentioned in previous sections, the processing dictionary transformed from the *general purpose* dictionary (MTF dictionary) allows us to combine declarativity of the dictionary description and flexibility of the translation process. However, the dictionary format transformation to realize this is not simple. In particular, the transformation

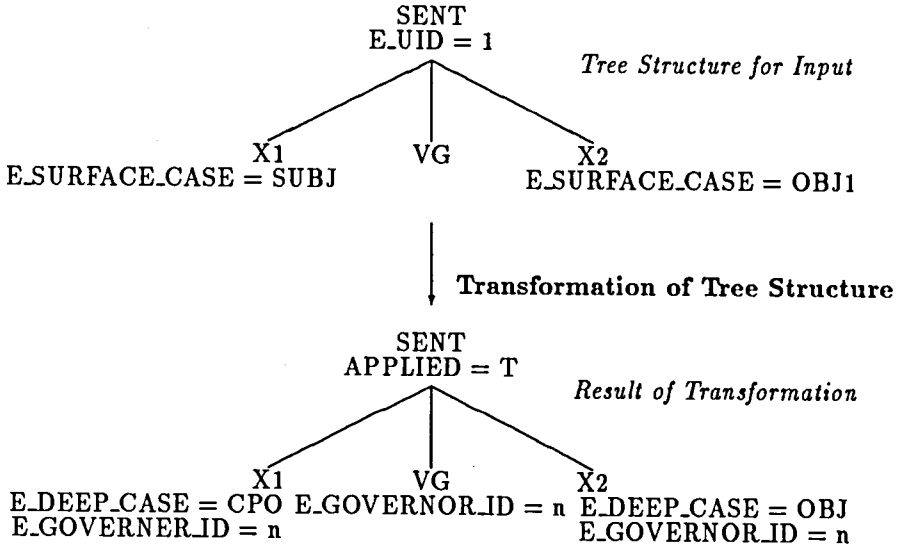


Figure 5.9: Tree Transformation by Dictionary Rule for English Analysis

from the declarative MTF dictionary to the grammatical rules of the rule-format dictionary is not easy. Of course, a program to generate grammatical rules from the MTF dictionary could be written in a conventional general purpose programming language such as LISP. But, it is difficult to maintain such a program, should the format of the MTF dictionary or the rule-format dictionary be changed. Therefore, we need special software tools for specifying the transformation process so that *future* modification and extension of the dictionary system may be supported easily.

The dictionary transformation can be divided into the following 2 phases:

1. extracting information from the MTF dictionary

2. filling the extracted information into specific expressions for the translation process (grammatical rules)

By dividing the transformation into these two phases, only the program for phase 1 needs to be modified when the MTF dictionary is changed. When the specification of the processing dictionary is modified, only the program for phase 2 needs to be updated. Independence between the MTF dictionary and the processing dictionary is increased

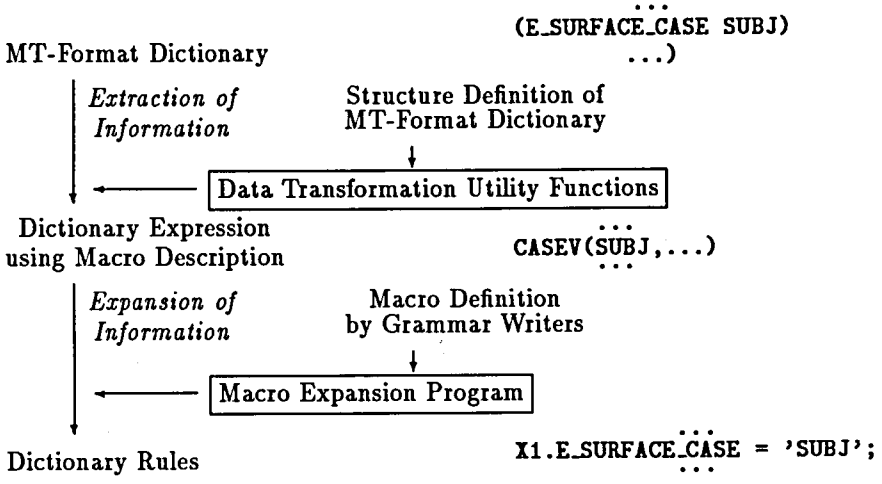


Figure 5.10: Flow of Transformation of Dictionary Rules

by this separation.

As software tools for the dictionary transformation, 'data transformation utility functions' and a 'macro language' have been developed for phases 1 and 2 respectively.[5] As shown in figure 5.10, the MTF dictionary is first transformed into an intermediate dictionary expression 'dictionary expression using macro description,' an example of which is shown in figure 5.11. This expression only contains information necessary to generate the rule-format dictionary (for example, SUBJ and CPO) which is extracted from the MTF dictionary. This intermediate expression is then macro-expanded into the rewriting rules which are the final result of the dictionary transformation.

In the following section, the software system for dictionary transformation is described using the example of generating the English analysis dictionary rule from a case-frame description of the verb 'affect.'

[5]In this section, tools to express the transformation process are explained. Tools for dictionary maintenance to execute the transformation process are discussed in section 6.4.1.

```
1  ANARV(affect,1,1,            /* Info. about the name of */
                                /* Rewriting Rule          */
2  CASEV(SUBJ,CPO,OBJ1,OBJ))    /* Info. about Case Frame  */
```

Figure 5.11: Example of Dictionary Expression using Macro Description

5.6.1 Data Transformation Utility Functions

The data transformation utility functions consist of two components [Nakamura84a]:

1. definition of property names in the MTF dictionary and operations for their values (DEF-MTF-CONV).

2. function to execute LISP programs according to the definition (MTF-CONV).

The program to generate a 'dictionary expression using macro description' from the MTF dictionary is easily written using these two components:

1. defining the way to extract information from the MTF dictionary according to the operation defined by the definition (1), and

2. writing the LISP program which calls the function (2).

Since the definition of the format of the MTF dictionary is independent of the program to use this definition, it is easy to modify the program when the format or the contents of the MTF dictionary is changed.

A part of the definition and a LISP program to generate the 'dictionary expression using macro description' for an analysis rule of the English verb is shown in figure 5.12. The definition is written according to the format (lines 2-3 in figure 5.12 (a)):

(<property name> <LISP program>...).

In this example, the lexical data whose property names are E_UID and C-FRAME are the targets of the operation. When these data are in the MTF dictionary, this LISP program is executed in an environment such that pre-defined global variables (?V and ?VS) are bound to the property values in the MTF dictionary. Therefore, the LISP program can use these global variables to generate the expression required. Note that ?V

```
1  (DEF-MTF-CONV ANAV-USAGE NIL NIL NIL      ; Definition name is
                                             ; ANAV-USAGE
2    ((E_UID     (SETQ !E_UID   ?V)  NIL)    ; Property Name and
                                             ; Operation for it
3     (C-FRAME   (SETQ !C-FRAME ?VS) NIL)
4     ))
```

(a) Definition of the data extraction from MT format.

```
1  (DEFUN WRITE-ANARV (E_LEX USAGE)          ; Program for generating
                                             ; Rewriting Rule
2    (LET ((CNT 1) (!E_UID "") (!C-FRAME NIL))
3      (LOOP (UNLESS USAGE (EXIT))
4        (MTF-CONV (POP USAGE) 'ANAV-USAGE)  ; Extraction of the data
5        (FORMAT "ANARV(/C,/C,/C,/N" E_LEX CNT E_UID)
6        (WRITE-CASEGV !C-FRAME)             ; Generation of
                                             ; Case Frame part
7        (INCR CNT 1))))
```

(b) LISP program for the rule generation.

Figure 5.12: LISP Program to Generate Macro Description from MT Format Dictionary (part)

is used for values which do not have internal structure, like E_LEX (English Lexical unit), and ?VS for values which do, like C-FRAME. The definition of figure 5.12 (a) extracts the value of E_UID and C-FRAME from the MTF dictionary, and assigns (SETQ) them into the variables !E_UID and !C-FRAME respectively, which are then used by the LISP program.

The program to generate a 'dictionary expression using macro description' in figure 5.12 (b) uses the definition ANAV-USAGE by calling the utility function MTF-CONV (line 4 in figure 5.12 (b)), and extracts the required data from a part of the MTF dictionary (a USAGE part of figure 5.5, lines 6–21). This means that execution of the function MTF-CONV assigns the values of E_UID and C-FRAME, which are in the USAGE part of the MTF dictionary, to the variables !E_UID and !C-FRAME respectively. After these assignments, the function FORMAT in line 5 writes E_LEX (head word), the sequence number of USAGE (CNT), and E_UID to a file. Then, the program writes the expression for the case frame part by calling another generation program WRITE-CASEV with the value of C-FRAME. The program WRITE-CASEV also extracts data according to the operation defined by DEF-MTF-CONV and writes a required expression.

Table 5.1: Format of Macro Language (part)

DEFINE(*string-0* (*arg-0*, *arg-1*, ...), *string-1*)	Replaces *string-0* with *string-1*. When *arg-n*'s are specified, they are also replaced.
IFELSE(*string-0*, *string-1*, *string-2*, *string-3*)	When *string-0* is equal to *string-1*, this form is replaced with *string-2*. Otherwise, it is replaced with *string-3*.

This program generates the 'dictionary expression using macro description' in figure 5.11 from the MTF dictionary in figure 5.5. In figure 5.11, line 1 is generated from line 4 in figure 5.5 and line 2 is generated by the function **WRITE-CASEV**. **ANARV** and **CASEV** in figure 5.11 are macro definition names which will be explained in the next section, and **SUBJ** and **CPO** are extracted from the (c) part of the MTF dictionary in figure 5.5.

5.6.2 Macro Language

The macro expansion program substitutes character strings according to a macro definition. The dictionary expression with macro description is expanded to GRADE grammar rules according to the macro definitions written by the grammar writers of the translation system. The grammar writers only need to write a macro definition which puts the data extracted from the MTF dictionary into a template of the grammatical rule expression.

The format of a macro definition is shown in table 5.1 [Nakamura84a]. For example, by defining a macro PAT as

```
DEFINE(PAT(X,Y),MC;%(X Y);),
```

when input data to the macro expansion program contains a string:

```
PAT(ADJ,NP),
```

it is replaced with the string:

```
MC; %(ADJ NP);.
```

```
1  DEFINE(CASEV                    /* Macro Name for English Analysis */
2          (!ESC1,!EDC1,           /* Parm. for First Case            */
                                   /* (Deep and Surface Case)         */
3           !ESC2,!EDC2,),         /* Parm. for Second Case           */
                                   /* (Deep and Surface Case)         */
4  MC; \%((SENT CASES));
5     SENT.E_UID = '!EUID';
6     CASES: %(X1 VG X2); X1,X2:DISORDER_SKIP( %( X ) );,
7     X1.E_SURFACE_CASE = '!ESC1';  /* Constraint for Surface Case */
8     X2.E_SURFACE_CASE = '!ESC2';

9  CR; **;
10    X1.E_DEEP_CASE <= '!EDC1';        /* Assignment of Deep Case */
11    X2.E_DEEP_CASE <= '!EDC2';
12    X1.E_GOVERNOR_ID <= VG.E_NODE_ID;
13    X2.E_GOVERNOR_ID <= VG.E_NODE_ID;
14    SENT.APPLIED <= 'T';
15 )
```

Figure 5.13: Example of Macro Definition to Generate English Analysis Rule for a Verb

Also, the expression

```
IFELSE(ESC,,,X.E_SURFACE_CASE = 'ESC';)
```

is replaced by the string

```
X.E_SURFACE_CASE = 'OBJ';
```

when 'ESC' is a non-null string, like OBJ, otherwise, it is replaced with a null string and the whole expression 'IFELSE(...)' disappears. In this way, it is possible to express not only simple string replacement, but also conditional substitution. Therefore, grammar writers can write complex macro definitions, if necessary.

A part of the macro definition to generate a grammatical rule (for the English analysis) for a verb is shown in figure 5.13. CASEV in this example is a macro definition name to generate a condition part and a result part of the rewriting rule which transforms surface cases into deep cases. This definition expands the expression in figure 5.11 into the GRADE grammar rule of figure 5.7. In this case, the following substitutions are performed:

```
!EUID ⇒ 1, !ESC1 ⇒ SUBJ, !EDC1 ⇒ CPO,

!ESC2 ⇒ OBJ1, !EDC2 ⇒ OBJ.
```

Since these substitutions are specified by SUBJ and CPO in figure 5.11, each line in figure 5.13 is replaced with the corresponding lines in figure 5.7.

Note that this example is a version of the macro definition used in the Mu-project, simplified for each of explanation in this chapter. Since the real MTF dictionary contains more complex information as discussed in section 5.5, the real macro definitions contain a lot of conditional expansion.

Grammar writers have flexibility in using the information in the MTF dictionary by writing macro definitions suitable to their grammar. In addition to this flexibility, they can also easily update their system simply by modifying their macro definitions.

5.7 Summary of this Chapter

Dictionary database is one of the key componemts of a machine translation system, and development and maintenance of the dictionary database are difficuly. The difficulity is mainly caused by the size of the dictionary. Thus we need to clarify the way to develop and maintain the dictionary database to reduce the difficulity.

In this chapter, the utilization method for the dictionary database in the Mu project has been described. A system to maintain the dictionary with two levels has been developed: descriptions dependent on processing are separated from the description of the linguistic features of each word. This system gives us the following merits:

1. We can develop and maintain the dictionary *independent* of the processing system. We are able to modify the description of the dictionary easily.

2. It is possible to express the linguistic information in a *structured* declarative form, without a deterioration in processing flexibility. As the information is declarative, it is easy to write programs for maintaining the dictionary, such as a dictionary editor, and a format checker for its the contents [Katagiri85].

3. We can use the same dictionary for different purposes, such as sentence analysis and sentence generation.

To realize this system, data transformation utility functions and a macro language have been developed as software tools for dictionary format transformation. Using these tools gives us the following:

1. It is easy to generate the description required by the processing system from the dictionary.

2. It is possible to deal with various modifications of the whole system. If the format of the dictionary is changed, a program using the data transformation utility functions needs to be updated. On the other hand, if the processing system is modified, only macro definition need to be changed.

The dictionary database stored in the MT format is also clear and understandable for humans. This is important since it allows us to develop various software systems for the handling of large amounts of linguistic information as is necessary for future research in natural language processing.

Chapter 6

Configuration of GRADE System

6.1 GRADE Software Environment

To support the development of a machine translation system, a good software environment is nesessary as discussed in chapter 2: softwares to execute the translation, and tools for the development and maintenance of grammars and dictionaries. In this chapter, configuration of the GRADE system as the development environment of a machine translation system is presented.

The GRADE system consists of the following sub-systems as shown in figure 6.1:

1. GRADE translator and executor for GRADE grammar rules. This component is the main part of the GRADE system. The GRADE executor also contains dictionary look up system.

2. Tools for grammar development. To develop a large and complex grammars for a *practical* machine translation, grammar writers need to use various software tools for incremental improvement of the grammars. The GRADE system has tools for this purpose.

3. Tools for dictionary maintenance. Dictionaries are important components of a machine translation system. Their largeness and complexity causes problems for the dictionary development and maintenance. The GRADE system provides tools to support the development and maintenance of dictionaries.

The GRADE system has been implemented on a mainframe system and transported into several machines as follows:

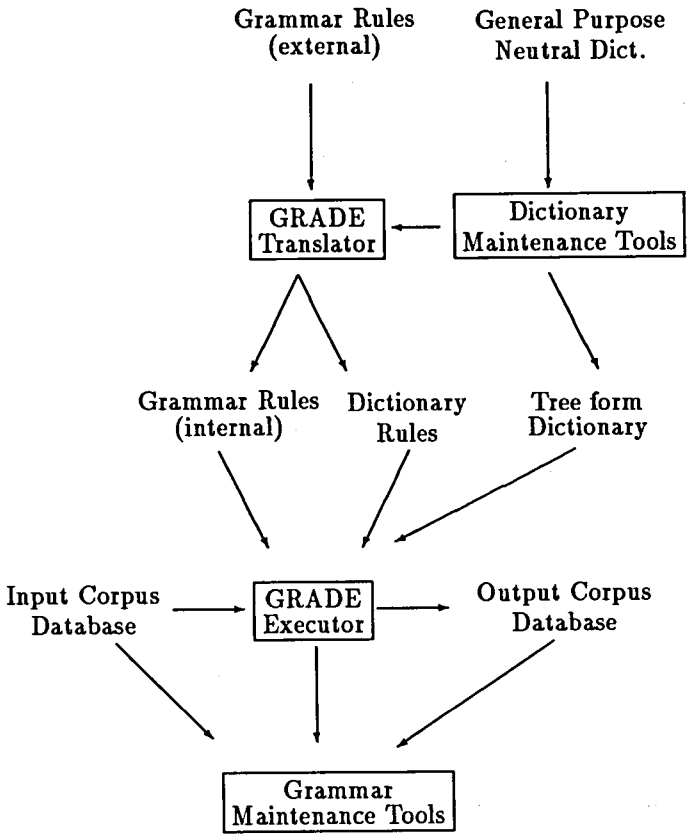


Figure 6.1: Configuration of GRADE Software Environment

- A UTILISP (University of Tokyo Interactive LISP [Chikayama81] [Chikayama83]) version on the FACOM M382/M780 (OS IV/F4 MSP) mainframe. This UTILISP has the additional functions of handling Kanji characters and direct access files (VSAM [Fujitsu83]) [Nakamura85a].

- A ZETALISP [Weinreb81] version on the Symbolics 3600 series (a partial implementation) [Nakamura84b].

- A KCL (Kyoto Common Lisp [Yuasa85]) version and an Interlisp-D version (a partial implementation), which have been transported by private companies that participated in the Mu project.

The size of the GRADE system is about 30,000 lines. In the rest of this chapter, the UTILISP version of the GRADE system is discussed.

6.2 GRADE Translator and Executor

The system configuration of the GRADE translator and executor is shown in figure 6.2. Grammar rules written in GRADE are first translated into internal forms, which are expressed by S-expressions in LISP. The internal forms are designed to be interpreted by LISP programs easily and effectively. This translation is performed by the GRADE translator. Then the internal forms are applied to an input tree by the GRADE executor.

6.2.1 GRADE Translator

The GRADE translator has three phases:

- Lexical analysis phase.

- Syntactic analysis phase.

- Internal form generation phase.

The lexical analysis routine reads grammar rules from a file and makes tokens. And it collects the tokens into 'the GRADE sentences', which are separated by ";". Then

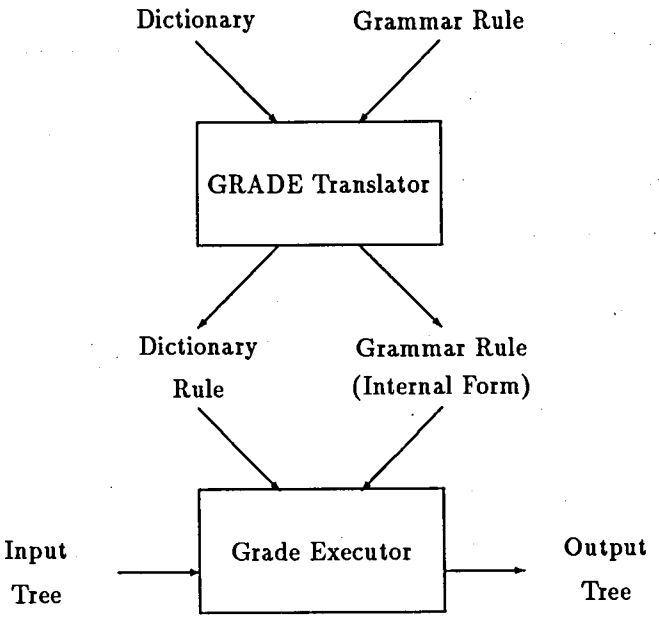


Figure 6.2: System Configuration of GRADE Translator and Executor

it executes a simple syntactic analysis of the sentences with an operator precedence method.

The syntactic analysis and internal form generation routine occur after the lexical analysis phase. One function of these routines is to find the relation between a tree structure and conditions of properties, which is written in the *matching condition part* implicitly for the readability of grammatical rules, and to decide the timing of property checking during a pattern matching.

In the rewriting rule shown in figure 6.3, the property condition of N node

```
N.E_NUMBER_TYPE = 'R1';
```

is written independently of the tree structure

```
%((NP #1 N #2));.
```

The syntactic analysis routine determines that the property condition check will be done after the pattern matching of N node, and the internal form generation routine makes

```
ZERO1.rr;
  directory_entry;
    owner(J.NAKAMURA); version(V01L01); last_update(83/03/06);
  matching_instruction;
    level(0,0); order(2,skip); tree;
  matching_condition;
    %((NP #1 N #2));
      N.E_NUMBER_TYPE = 'R1';
  creation;
    %((NP ART #1 N #2));
      ART.E_LEX <= 'NIL'; ART.E_SUBCAT <= 'ZERO';
  end_rr.ZERO1;
```

Figure 6.3: External Expression of Rewriting Rule

```
(rr rr_ZERO1
  (("owner       = J.NAKAMURA"
    "version     = V01L01"
    "last_update = 84/03/06"
    "translated  = 09/28/87 13:48:54"
    "source      = 'a51388.gen.grade(art)'")
   rrapplier
   (nil ((ART nil)) nil ((n001) (NP) (N) (#1) (#2)))
   (E_CAT (0 . 100) lbd (2 . skip) para nil)
   (matcher
     (!cont n000 ((!nop NP NP nil)))
     (!cont NP
            ((!anythings #1)
             (!nop N N
                   (!prop-check
                     ((ueq (pvar N E_NUMBER_TYPE)
                           (pvalue ((R1))))))
             (!nop #2 nil nil))))
   nil
   (creation
     (construct
       (prop-set
         ((uset ART E_LEX (pvalue (nil)))
          (uset ART E_SUBCAT (pvalue ((ZERO))))
          (user ART E_CAT (pvalue ((ART))))))
       (constpatt (NP ART #1 N #2))))))
```

Figure 6.4: Internal Expression of Rewriting Rule

the S-expression shown in figure 6.4:

```
(!nop N N
      (!prop-check ((ueq (pvar N E_NUMBER_TYPE)
                         (pvalue (R1))))))).
```

This internal form means that

- If there is a tree and the value of the property E_CAT (English CATegory symbol) in the root node is N,
  1. assign it to the variable N,
  2. get the value of the property E_NUMBER_TYPE of the tree assigned to N, and
  3. compare the value with R1.
- Otherwise, fail the pattern matching.

When the internal form is interpreted, it does not need to be tested by the program at the time of property condition checking.

Note that the internal form of grammar rules is stored in a rule *database* implemented with the VSAM file. The GRADE executor retrieves the internal form from the database on demand (see also section 6.3.2).

6.2.2 GRADE Executor

The internal form of grammar rules is applied to an input tree, which is an output of the morphological analysis program. This rule application is performed by the GRADE executor. The result of rule applications is sent to the morphological generation program.

The GRADE executor consists of three parts:

- Subgrammar network and subgrammar application routine.
- Rewriting rule application routine.
- Tree transformation routine.

The subgrammar network and subgrammar application routine is a toplevel function. It receives an input tree structure, calls the rewriting rule application routine by interpreting the internal forms of subgrammar network and subgrammar, and outputs the transformed tree structure. It uses a backtracking mechanism to support the para special node, which is explained in section 3.7.

The rewriting rule application routine traverses an input tree in accordance with the specification written in a matching instruction part, and calls the tree transformation routine.

The tree transformation routine consists of a pattern matcher, a program to execute a substructure operation part, and a program to make a transformed tree. This routine changes an annotated tree by interpreting the transformation part in a rewriting rule.

6.3 Tools for Grammar Development

In a *practical* machine translation system, we need a very large grammar to handle variety of natural language phenomena, because there are many general rules and a vast amount of (*heuristic*) rules specific to words, sentential pattern, and so on. To develop such a large grammar, many grammar writers and lexicographers must work cooperatively, in contrast to the development of a *model* system. Software tools for the grammar development should support this co-operative work.

Software tools for the grammar development are classified into two types:

- Tools for *dynamic* analysis: these tools are mainly used interactively at the translation test phase to inspect the flow of a translation process in detail.

- Tools for *static* analysis: we need several tools such as a tool to find linguistic phenomena from a corpus database, a tool to inspect the current state (version) of the grammar development, and so on.

In the following sections, we discuss the tools designed and used to develop the Mu translation system.

6.3.1 Tools for Translation Test

Software tools used for the translation test are similar to *debugging tools* of a typical programming system: a *tracer* to examine an execution process, a *breaker* to halt the execution at a specific point, a *stepper* to run a program step-by-step, an *inspector* to examine a data structure in detail, a *disassembler* to show the external form of an assembled program, and so on. Software tools analogous to these are useful for the development of a grammar. However, there are several differences between them, because:

- The data structure to be handled by tools is restricted to a *tree* structure. Tools should be specialized to handle a *tree* structure.
- Users of tools are not experts in programming. The user interface therefore should be designed to be simple and easy to use.

Keeping these points in mind, the following tools have been developed to support the grammar development of the Mu project:

- Tree Printer
- Tracer
- Breaker
- Disassembler
- Interactive Inspector (TBX)

Tree Printer

The data structure which tools need to handle is an *annotated tree* structure in the GRADE system. *Tree Printer* is a tool which is designed for grammar writers to inspect the intermediate tree structure in detail. Figure 6.5 shows an example of the output from the tree printer.[1] In this figure, a grammar writer wants to check the details of

[1] In figures, Japanese words, such as SHINPO in figure 6.5, are expressed with Romaji. In the implemented system, however, Japanese words are expressed with Kana and Kanji combination.

```
ENTER TBX COMMAND:
(SHOW ASSIGN EDIT TEST USE EXIT HELP)>SHOW TREE IN
SOME PROPERTY NAME OR C/R --> E_DEEP_CASE E_SURFACE_CASE

?  <?>  <>  <>                                                    1
|--V  <increase>  <>  <>                                         2
   |--N  <advance>  <CAU>  <A_OTHERS>                            3
   |  |--N  <?>  <AGT>  <SUBJ>                                   4
   |     |--N  <electrical measurement>  <PARAELEMENT>  <>       5
   |     |  |--?  <?>  <>  <>                                    6
   |     |     |--ADJ  <electrical>  <>  <>                      7
...
         |--N  <ship>  <>  <>                                    20

NODE ID/A/T --->3
3      (($midashi-go (shinpo)) ($koubun-hinshi (meishi))
        ($shinsou-kaku (genin-riyuu)) (SEQ (2052)) (E_CAT (N))
        (E_LEX (advance)) (E_UID (1)) (FC_V_TRANSFER (T))
        (E_TOP_POSITION (T)) (E_PREP_LEX (by)) (E_DEEP_CASE (CAU))
        (E_NOOBCASE (T)) (E_SURFACE_CASE (A_OTHERS)) (E_SURFCASE_DONE (T))
        (E_SEM (PS)) (E_SUBCAT (COM)) (E_UID (1)) (E_NUMBER_TYPE (C))
        (E_POSITION (NO_MARK)) (E_VERB_RELATED (/-)) (E_ADVERB_LIKE (/-))
        (E_N_TO_PP (OBJ)) (E_CONVERSION (V)) )
```

Figure 6.5: Example of Tree Printer

an input tree structure for the English generation grammar. This is expressed with the command:

```
SHOW TREE IN.
```

This is a command to the interactive inspector TBX, which is explained in a later section.

An *annotated tree* in GRADE contains various information in its nodes: syntactic information, semantic information, information to control the translation process, etc. It is not useful to display all information at once, because the output becomes too complicated for a grammar writer. Therefore, the tree printer has two options: a selection of properties in a tree form output, and interaction with the user to display all information of a specified node.

In a tree form output, values of E_CATS (English category symbol) and E_LEX (English lexical unit) in nodes are always displayed. These two properties are defaults for the English generation grammar of the Mu project. In addition to these defaults, a grammar writer is allowed to specify additional property names. For example, the line in figure 6.5:

```
SOME PROPERTY NAME OR C/R --> E_DEEP_CASE E_SURFACE_CASE
```

specifies to output E_DEEP_CASE and E_SURFACE_CASE additionally.

The printer first display the outline of a tree structure with temporal node-ID numbers for its nodes. When the grammar writer wants to check the details of the nodes, he can ask the tree printer to display them with these temporal node-ID numbers. Then, the tree printer outputs all information stored in the specified node as shown in the last part of figure 6.5. In this example, the details of the node 3, which contains the information about the noun '**advance**', is displayed. From this display, we see the (default) preposition (E_PREP_LEX) of this noun is '**by**', for instance.

Tracer

The grammars of the Mu project have been developed with the idea of a *procedural grammar* [Tsujii84] and their tree-to-tree transformation process of an input sentence has a similarity to the execution of a program. Therefore, tracing the tree-to-tree transformation process of the grammars is useful for a grammar writer to inspect the grammars.

Figure 6.6 shows the example of an output from the tracer of the GRADE system. The tracer outputs two types of information about the execution of grammars:

- Names of rules which are applied to a tree structure and their result: applied or not. The message:

  ```
  + SGN_G_N_MAIN
  ```

  shows that a SubGrammar Network G_N_MAIN started the application to a tree structure and the message:

```
 + SGN_PRE_MAIN_GENERATION
   + SGN_G_CHECK_ROOT_AND_PUNC
     + SGN_GENERATION_MAIN
...
       + SGN_GENERATION
         + SGN_G_V_MAIN
           + SGN_G_V_DICT_POST
             + SGN_G_N_MAIN
               + SGN_G_N_N_GENERATION
                 + SGN_GENERATION
                   + SGN_G_N_PARA
                     + SGN_GENERATION
                       + SGN_G_N_MAIN

 *** TREE(CONTINUE) ******
               --------UNIT : SGN    RULE : SGN_G_N_MAIN ***

N  <electrical measurement>  <PARAELEMENT>  <>                    1
|--N  <electrical measurement>  <PARAELEMENT>  <>                 2
|  |--?  <?>  <>  <>                                             3
|     |--ADJ  <electrical>  <>  <>                               4
|     |
|     |--N  <measurement>  <>  <>                                5
|
|--DEL  <?>  <>  <>                                              6

                     - SGN_G_N_MAIN  APPLIED (319 MS)

 *** TREE(CONTINUE) ******
               --------UNIT : SGN    RULE : SGN_GENERATION ***

NP  <?>  <PARAELEMENT>  <>                                       1
|--N  <electrical measurement>  <PARAELEMENT>  <>                2
   |--?  <?>  <>  <>                                            3
      |--ADJ  <electrical>  <>  <>                              4
      |
      |--N  <measurement>  <>  <>                               5
```

Figure 6.6: Example of Tracing and Printing a Tree

```
- SGN_G_N_MAIN APPLIED (319 MS)
```

shows that its application was successful in transforming the tree structure.

These messages also show a *calling sequence* of the rules. In figure 6.6, for example, one subgrammar network PRE_MAIN_GENERATION calls other subgrammar networks G_CHECK_ROOT_AND_PUNC, etc.

Note that this message also contains the execution time (CPU time) from the start of whole processing. In this case, it took 319 milliseconds for the execution.

- Tree structures. When a rule transforms a tree structure, the tracer outputs the tree with the tree printer. In this example, in figure 6.6, the subgrammar G_N_MAIN modified a tree structure from a 'dependency structure'-like tree (top node is 'N') into a phrase structure tree (top node is 'NP'). This allows grammar writers to check the grammar works correctly or not.

Using the tracer, a grammar writer roughly locates the rule which causes a problem for translation. After that, he may use the breaker, which is explained in the next section, to inspect the grammar in more detail.

Breaker

When a grammar writer wants to inspect the behavior of a rule in detail, the *breaker* is useful. The breaker allows the halting of the execution of the GRADE system before and/or after the application of a specified rule. In addition to this, it has an option to stop the execution when the system assigns a value to a specified property. Figure 6.7 shows an example of this interaction with a grammar writer.

In this example, a grammar writer wants to stop the execution before and after the application of the rule: G_N_MAIN. This is specified at the beginning part of this session. After this specification, GRADE system starts the execution and outputs tracing messages until the system tries to apply the specified rule(s) and stops the execution.

When the system stops the execution at the break point, a grammar writer may use TBX (Tracer/Breaker eXtension). The end of figure 6.7 shows this interaction. TBX

```
+++ TRACER OPTION SETTING +++
 T - TRACE        B - BREAK POINT        P - PROPERTY ASSIGN BREAK
 C - CONTINUE     TBX - TB EXTENSION
-->B
++ BREAK POINT ++
ENTER BREAK MODE(B I O), UNIT NAMES, AND Y(IF YOU SEE DEFAULT MODE).

==>B SGN_G_N_MAIN ⟵ to specify rule names to stop the execution

 + SGN_PRE_MAIN_GENERATION
...
                        + SGN_G_N_MAIN

++ BREAK ++  SGN_G_N_MAIN JUST STARTED +
  P - TREE PRINTER   L - TREE LIST  V - VARIABLE  N - NO
-->
+++ RESTART? C/R:YES, S:SAVE & RESTART, X:STOP, T:TRACE, B:TBX

-->B ⟵ to enter TBX mode

--- TBX BREAKER FOR SGN_G_N_MAIN IN BREAK-START (V06L26)...
```

Figure 6.7: Example of Break Point

has a lot of operation menus and its details are explained in a later section. Here, the output of one operation

```
SHOW STATUS
```

is shown in figure 6.8. With this command, a grammar writer checks the current status of the break point. He can check, for example, the name of the current corpus database (`TREE VSAM : 'A50650.JETREE.J320.VSAM'`), the name of the rule stopped the execution (`BREAK-POINT RULE : SGN_G_N_MAIN`), and so on.

Disassembler

At execution time, GRADE grammar rules are expressed in the internal structures (LISP S-expressions) as discussed in section 6.2.1. For the system programmer of GRADE, it is not difficult to understand the *meaning* of a rule from its internal structure such as shown in figure 6.4. However, it is not so easy for grammar writers to do so. We need a disassembler, which re-translates an internal structure into an external expression.

```
--- TBX BREAKER FOR SGN_G_N_MAIN IN BREAK-START (V06L26)...
ENTER TBX COMMAND:
(SHOW ASSIGN EDIT TEST USE EXIT HELP)>SHOW STATUS

---------- TBX STATUS (08/28/87 17:26:45) ---------
PROCESS            : GTEST                          SENTENCE NO : E82060001_3_1
TREE VSAM          : 'A50650.JETREE.J320.VSAM'
BREAK-POINT TYPE   : BREAK-START                    APPLIED : ?
            RULE   : SGN_G_N_MAIN                   FUNC    : SGN:APPLIER
TREE IN            : YES                            OUT     : NO
SGN-NAME           : SGN_G_N_MAIN
SG-NAME            : SG_GENERATION
RRNAME             : RR_G_N
SEGMENT FRAME      : (SGN_G_N_MAIN SG_G_N_PRE SG_GENERATION
                      SG_G_V_DICT_POST SGN_G_V_DICT_POST SG_G_V_DICT
                      SGN_G_V_MAIN SG_G_V_PRE)
SEGOUT INHIBIT     : NO
EDITOR MODE        : USE
RULE PRINT MODE    : PP                             PRINT LEVEL : 1
     WORD TABLE    : YES
     RT LEVEL      : 4
LOG DSNAME         : NIL
    MODE           : NO                             TERMINAL    : YES
```

Figure 6.8: Status of Break Point

```
--- TBX BREAKER FOR SGN_G_N_MAIN IN BREAK-START (V06L26)...
ENTER TBX COMMAND: (SHOW ASSIGN EDIT TEST USE EXIT HELP)>SHOW RULE FORM *

 /* GRADE DISASSEMBLER V02L10 */
 G_N_MAIN.SGN;
   DIRECTORY_ENTRY;
     /* OWNER       = Y.FUKUMOCHI          VERSION     = V03L01 */
     /* LAST_UPDATE = 85/02/18             TRANSLATED  = 03/13/85 16:02:23 */
     /* SOURCE      = 'A51388.GEN.GRADE(NPGEN)' */
   ENTRY; N_MAIN_START;
   NETWORK;
     N_MAIN_START
              :G_N_DIC.SG;
     N_DET    :G_N_DET.SG;
     N_ADJ    :G_N_ADJ.SG;
     N_MAIN   :G_N_MAIN.SG;
     N_N      :G_N_N.SG;
     N_ADJ_EMB
              :G_N_ADJ_EMB.SG;
     N_V      :G_N_V.SG; EXIT;
 END_SGN.G_N_MAIN;
```

Figure 6.9: Output of Disassembler

Figure 6.9 shows an example of the output from the disassembler for the subgrammar network G_N_MAIN at which the system halts the execution. The command

```
SHOW RULE FORM *
```

makes the disassembler output the current rule. In addition to the GRADE source format, the disassembler displays:

- translated date and time (03/13/85 16:02:23)
- source file ('A51388.GEN.GRADE(NPGEN)').

This information is stored in the internal structure and is useful for the improvement of grammar rules, especially when grammar writers are working cooperatively.

TBX — Tracer/Breaker Extension

The software tools of the GRADE system have two interactive mode. One mode is very simple and a grammar writer does not have to know the details of the tools. In this mode, the system displays a menu each time the system performs an option of the *debugging* mode. If a grammar writer does not want to do anything special, he just pushes the *return-key*. Then, the system takes a default action. This mode is useful when a grammar writer is updating his own grammar rules, because he knows almost everything about his grammar rules and he only needs to see an intermediate tree structure at a specified phase.

On the other hand, a grammar writer sometimes need to inspect grammar rules in more detail, when he is updating grammar rules written by other grammar writers. In this case, he needs a more *sophisticated* interface to the software tools. Therefore the GRADE system supports another mode for the interaction with a grammar writer. This mode is called TBX (Tracer/Breaker eXtension).

TBX has many commands for grammar *debugging*, which are shown in figure 6.10. Commands are classified into four types:

- SHOW command to see various information such as tree structures (TREE, figure 6.5) and grammar rules (RULE, figure 6.9).

```
TBX HELP (V06L09, TBX VERSION IS V06L26):
 SHOW    STATUS | VAR | OPTION | PROCESS | INPUT
         TREE     IN | OUT
         RULE     FORM       * | RR | SG | SGN
                             NAME   <RULE-NAME>
                  COMMENT
                  DEFAULT    * | RR | SG | SGN
                             NAME   <RULE-NAME>
                  REFERENCE  * | RR | SG | SGN
                             NAME   <RULE-NAME>
                  BACKTRACE
 ASSIGN  TPR        MODE         <MODE-NAME>
                    SPL          C//R <PROP-NAME>...
         OPTION     TRACE        C//R <TRACE-PARM>
                    BREAK        SET        C//R  <BREAK-PARM>
                                 RESET      <RESET-PARM>
                    PROP-BREAK   SET        C//R <PROP-PARM>
                                 RESET      ALL
                    CONT-TREE    C//R <CONT-TREE>
         FLAG       SGSEG-OUT-INHIBIT-P     YES | NO
         RULE       RR | SG | SGN           <RULE-NAME>
         TREE       IN | OUT                <TREE-VAR>
         PROCESS    <PROCESS-NAME>
         TBX-MODE   EDIT           USE | PFD
                    SHOW-RULE      PP | SEX
                    WT-RULE        YES | NO
                    LEVEL-RULE     <NUMBER>
                    RT-LEVEL       <NUMBER>
         LOG        START <DS-NAME>
                    END | ON | OFF
                    TERMINAL       ON | OFF
 EDIT      TREE     IN | OUT
           RULE     * | RR | SG | SGN
                    NAME <RULE-NAME>
 TEST      * | SG | SGN
 USE       EDIC | EDIC-TRANS | XREF-TRACE
 EXIT | HELP | <SEXPR?
 X         <TSS-COMMAND>
```

Figure 6.10: TBX Commands (help message of TBX)

- ASSIGN command to set several options of the *debugging* mode.
- EDIT command to modify a tree structure and grammar rules.[2]
- Other miscellaneous commands. For instance, the command (X) to execute a TSS command (a command of the operating system) such as to LIST a source file.

When a grammar writer needs to know details of the grammar rules, TBX is of great value.

6.3.2 Tools for Static Analysis

We need two types of tools for *static* analysis to develop the grammar systems. One is for grammar rules themselves, because the number of grammar rules becomes larger and larger in proportion to progress of the system development and they cannot be maintained manually. The other tool is for text databases to find new linguistic phenomena and problems of the grammar rules. The remainder of this section describes 4 tools which was developed for the Mu project.

Comment of Rule Database

Grammar writers often *version up* the grammar rules concurrently. This concurrent update sometimes causes inconsistencies in the grammar. Therefore, the GRADE system allows grammar writers to attach a comment to a rule database[3] as shown in figure 6.11.

The last line of this figure, for example, shows that

> A grammar writer, KUME, modified rules relating to article handling on March 22 (line 20).

Such a comment is useful for co-operative development of the grammar rules.

[2]The current version of TBX does not allow a grammar writer to modify external forms of the grammar rules interactively.

[3]The internal structures of GRADE grammar rules are stored in a direct access file which is implemented on VSAM [Fujitsu83]. Therefore, we call it as a rule database.

```
VSAM RULE FILE ('A52801.RULE.G.VSAM') UPDATE COMMENTS:
1  : 02/20/85 10:26:51 - V_N RULE (E_SURFACE_CASE) <KUME>
2  : 02/20/85 15:28:40 - MORCOMP (MONTH YEAR) <KUME>
3  : 02/20/85 22:07:15 - (PAREN) PARAROOT GEN ... REVISED ...<FUKU>
4  : 02/21/85 10:44:10 - (NV)(NUMBER) ... <FUKU>
...
16 : 03/20/85 14:14:39 - NP JOINT RULE <KUME>
17 : 03/20/85 17:21:36 - (CAUSE) SUBJECTIVIZE E_SEM REVISED ...<FUKU>
18 : 03/20/85 17:37:35 - (NUMBER) DICCOMP NO SHORI REVISED ... <FUKU>
19 : 03/20/85 20:08:05 - (REDCOP) REVISED ... <FUKU>
20 : 03/22/85 14:14:26 - ARTICLE RULE (DEFT_ART) <KUME>
```

Figure 6.11: Comments of Rule Database

Retrieval Program for Grammar Rules

The difficulty to find the exact rules which a grammar writer should inspect for the improvement of the system increases in proportion to the size of the grammar. Hence we need a retrieval program for the grammar rules. Therefore, a software tool for displaying a *static* calling sequence of the rules is supported by GRADE system.

Figure 6.12 shows the part of a calling sequence of the rule SGN_JAG (Japanese Analysis Grammar), which is a top level subgrammar network name for the Japanese analysis grammar of the Mu project. This output[4] tells us:

- Japanese Analysis grammar consists of the subgrammar networks (SGN) I, A, and D.
- The subgrammar network A, which is a main part of the analysis grammar, calls several subgrammar networks (e.g. J_BUN_TO_BUN_NI_WAKERU, which means "to separate an input sentence (compound sentence) into simple sentences.")
- Each subgrammar network consists of several subgrammars (SG).
- The total number of the displayed rules is shown in the last part of the listing. 37 subgrammar networks, for example, are listed in this output.

[4]This tool does not display *all* rules, and the level for displaying is specified by TBX command. In this case, 3 levels are displayed.

```
ENTER TBX COMMAND:
(SHOW ASSIGN EDIT TEST USE EXIT HELP)>SHOW RULE REFERENCE NAME

NAME>SGN_JAG
+ SGN_JAG IN PROCESSING +

SGN_JAG
 - SGN_I
    ...
 - SGN_A
    - SGN_J_BUN_TO_BUN_NI_WAKERU
       - SG_J_BUN_WO_WAKERU
    - SGN_J_KAKKO_NO_SYORI_1
       - SG_J_TOKUSHU_KIGOU_NO_SYORI
       ...
    - SGN_J_RENYOUCHUSHI
    ...
    - SGN_DECISION_SCOPE_OF_NOUN_PHRASE
       - SG_SEARCH_CANDIDATE_OF_NOUNS
       ...
    - SGN_J_GCS
    ...
    - SGN_ASPECT_ANALYSIS_1
       - SG_ASPECT_MAIN
    - SGN_J_BUNIMI
       - SG_J_BSZ_BUNBUN_KA
       - SG_J_RENYOU_BUNBUN_KA
    - SGN_TENSE_ANALYSIS_1
       - SG_TENSE_MAIN
       ...
 - SGN_D
    ...

      UNIT NO KOSU  -----  SGN      = 37
                           SG       = 60
                           RR       = 24
                           CALL-DIC = 0
```

Figure 6.12: Static Calling Sequence of Rules (part)

In contrast to the output from the tracer, this output contains *all* rules which have a possibility to be called from the rule. This is useful when some rules do not call other rules correctly.

Database of Source Sentences and Their Translations

To improve grammars, especially the transfer grammar, grammar writers need to compare translations by humans with their source sentences. For this purpose, the GRADE system supports a retrieval program to find sentences and their translations stored in a database with two keywords (character level) from English or Japanese. Figure 6.13 shows the example of this sentence database retrieval.

This example shows the translations (English) which contain the word '**describe**' and their source sentences (Japanese). This result allows grammar writers to examine what kind of sentential constructions are translated into the word '**describe.**' Grammar writers use this tool to find such an interlingual relation for improving the grammar rules.

Corpus Database

In a certain stage of the development of a translation system, we need to intensively examine the *corpus* (sample sentences). Grammar writers repeat the trial translations by one version of their grammar rules to improve the grammar. They need to know which sentences are correctly analyzed, transferred, and generated, and to konw when sentences are processed. A software tool to support this process is necessary for the development of a machine translation system.

In the GRADE system, sample sentences and their internal tree structures (results of analysis, transfer, and generation phases) are maintained in a corpus database. This corpus database also contains information related to translation tests. Figure 6.14 shows the information stored in the corpus database. First column shows the id-number of a sentence. The rest of the line indicates the date and time of analysis (AOUT), transfer (TROUT) and generation (GOUT). Last 2 lines show how many sentences are *correctly*

```
ENTER KEYWORD IN ENGLISH(JICST) OR <CR> FOR QUIT:'describe'
ENTER SECOND KEYWORD OR <CR>:
:
DATA BASE= ENGLISH(JICST)                              15.04.29, 87/01/10
:
FIRST KEY='describe'
FOUND#=          1   ABST#='E82060002'      SENT#='    3';
  rapid growth in the construction of schools and housing,
  electrification, high-rise collective residential buildings,
  construction of public buildings including local government offices,
  waste water processing plants, problems of refuse incineration
  plants, tunnels, free ferries, control of lighting energy and
  properties, energy saving activities, etc. are >>>described from a
  wide angle.
:
  急速な学校建設や住宅建設。
:
FOUND#=          2   ABST#='E82060003'      SENT#='    2';
  refeering to electrical rotaing machines, techniques of analyzing
  electromagnetic fields, and integrated electro-machanical design
  techniques, etc., this article >>>describes modest but steady
  progress and discusses the educational problem of it being necessary
  to educate, train, and recruite engineers who have creativity in not
  only the above-mentioned fields but also in long-established fields
  in the electrical industries.
:
  回転電気機械，電磁界解析技術，電気と機械の総合設計技術などを例に，地味
  ではあるが着実な進歩をのべ，教育の問題として，上記に拘らず電気産業の長
  く確立された分野にも創造力に富む技術者を補充し教育訓練する必要性を論ず
  る。
:
FOUND#=          3   ABST#='E82060005'      SENT#='    1';
  this article >>>describes an ultrasonic welding system and process as
  one of such techniques.
:
...
```

Figure 6.13: Example of Sentence Database Retrieval

```
KEY              :AOUT                TROUT                GOUT
...
E82060003_2_1    :06/10/85 17:54:11   06/15/85 10:00:15    06/15/85 10:02:13
E82060003_3_1    :06/10/85 18:02:17   06/15/85 10:00:47    06/15/85 10:02:59
E82060003_4_1    :06/15/85 09:02:06X
E82060004_2_1    :03/04/87 11:48:38   06/10/85 22:13:50    06/10/85 22:14:42
E82060004_2_4    :
E82060004_3_1    :06/10/85 21:57:04   06/10/85 22:13:58    06/10/85 22:15:04
E82060004_4_1    :06/10/85 22:00:01   06/10/85 22:14:10    06/10/85 22:15:58
...
-----------------
SUCCESS          :   18/   21 ( 85%)     18/   21 ( 85%)     18/   21 ( 85%)
ESCAPED          :    2/   21 (  9%)      0/   21 (  0%)      0/   21 (  0%)
```

Figure 6.14: Displaying the Information about Corpus Database

processed viewed from the grammar rules themselves.[5]

The mark 'X' after the date and time (e.g. analysis date and time '**06/15/85 09:02:06**' of the sentence E82060003_4_1) shows that the analysis grammar failed in the processing this sentence, which is detected by the grammar rules themselves. Grammar writers need to inspect sentences marked 'X' especially to find problems with the grammar rules.

This corpus database is also used to output source sentences and their translations. Figure 6.15 shows the output from the corpus database.

6.4 Tools for Dictionary Maintenance

Dictionaries are a key part of a *practical* machine translation system. The large size of the dictionaries of a *practical* system cause problems for the development and maintenance of dictionaries. In addition to this, because the dictionary system of the Mu project uses the idea of a *general purpose* and *neutral* dictionary as discussed in chapter 5, lexicographers of the Mu system need more *sophisticated* dictionary maintenance tools than a system that uses *simple* dictionaries. In the following sections, maintenance tools developed for the Mu project are discussed:

[5]Each grammar of the Mu system contains 'rules' to check the formal structure of the result.

NO. 2: E82060001_3_1 (10/29/85, 10/29/85, 10/29/85)
電気計測法，データ処理，自動化機器の進歩で自動化船が増加した。

Advances in electrical measurement, data processing and automatic equipment increased the number of automated ships.

NO. 3: E82060001_4_1 (06/10/85, 11/01/85, 11/01/85)
航海法も INMARSAT,MARSAT システムになった。

The navigation also became INMARSAT and MARSAT systems.

NO. 4: E82060001_5_1 (06/10/85, 06/10/85, 06/10/85)
電気推進船が多くなる傾向で，ディーゼル，ガスタービン，ターボ発電の
いづれかを使用する。

Electrically power-propelled ships use any of diesels, gas turbines and turbogeneration of many.

NO. 5: E82060001_6_1 (06/10/85, 06/10/85, 06/10/85)
また併列発電時の短絡問題の解決，ディーゼル推進船で推進軸直結発電法の開発，ダイナミックポジショニング用原動機の研究開発など 83 の文献調査により紹介。

Then, the introduction is made by references of 83 such as the development of the propeller shaft direct coupling power generation method and the research and development of dynamic positioning motors by solutions and diesel-powered ships of short circuit problems at the time of the parallel power generation.

Figure 6.15: Input Sentences and their Translations

1. A dictionary format transformation system to support the idea of a *neutral* dictionary.

2. A dictionary retrieval system to find and summarize the contents of dictionaries.

6.4.1 Dictionary Format Transformation Tool: Edic-Trans

The feature of the dictionary system of the Mu project is a *general purpose* and *neutral* dictionary database which is independent of dictionaries for a translation processing. The dictionary used in processing is transformed from the *neutral* dictionary. This transformation is not simple for lexicographers and grammar writers, because each dictionary needs a different transformation. For instance, a verb dictionary for English *generation* needs a different transformation from a verb dictionary for English *analysis*. Therefore, a software tool for the dictionary format transformation has been developed and it is called as Edic-Trans (Editor and TRANSformer for DICtionary).

Menu System

Edic-Trans system is a menu-based system, because this system is designed to be used by lexicographers, grammar writers, who are not experts of computer systems, and software engineers who maintain the dictionary system as a whole. The menu system of Edic-Trans has two modes:

- Dictionary selection mode.

- Operation mode.

A user of Edic-Trans first selects a dictionary which he wants to maintain, then executes operations for it.

When a user starts Edic-Trans system, he is first asked about the dictionary selection as shown in figure 6.16. There are 8 types of dictionaries at this dictionary selection mode in the Mu project. Note that there are 10 dictionaries in the Mu system:

- 4 *neutral* dictionaries: Japanese, English, J-to-E, and E-to-J.

- 6 dictionaries for *processing*:

```
>(EDIC-TRANS)
;EDIC-TRANS V01L10 (09/05/85) START...
SELECT DICTIONARY TYPE:
1 :SUPER                                2 :JAPANESE-DICTIONARY
3 :J-ANALYSIS-DICTIONARY                4 :JE-TRANSFER-DICTIONARY
5 :E-GENERATION-DICTIONARY              6 :ENGLISH-DICTIONARY
7 :E-ANALYSIS-DICTIONARY                8 :EJ-TRANSFER-DICTIONARY
9 :J-GENERATION-DICTIONARY
ENTER ITEM:5
SELECT DICTIONARY TYPE:
1 :SUPER                                2 :E-N-GEN
3 :E-V-GEN                              4 :E-ADJ-GEN
5 :E-ADV-GEN                            6 :E-DET-GEN
7 :E-PRON-GEN                           8 :E-PREP-GEN
9 :E-CONJ-GEN                           10:E-CARD-GEN
11:E-ORD-GEN                            12:E-ART-GEN
13:E-AUX-GEN                            14:E-UNIT-GEN
ENTER ITEM:3
SELECT TRANSFORMATION TYPE (LEVEL 0) FOR E-V-GEN:
1 :SUPER                                2 :EXIT
3 :LEVEL                                4 :DESCRIBE
5 :GET-MTF                              6 :LIST-SESSION-LOG
7 :CLOSE-ALL-FILES                      8 :MTF-VSAM-TO-SUMMARY
ENTER ITEM:
```

Figure 6.16: Dictionary Selection of Edic-Trans

- 3 for the J-to-E system: Japanese-analysis, J-to-E, and English-generation.
- 3 for the E-to-J system: English-analysis, E-to-J, and Japanese-generation.

However, *neutral* dictionaries and dictionaries for *processing* of the Japanese-to-English transfer and the English-to-Japanese transfer are the same, viewed from the dictionary transformation. Therefore, only 8 items appeared in the menu. In figure 6.16, a user selected the English generation dictionary (item 5) first and selected the verb dictionary in it (item 3, `E-V-GEN`).

After the dictionary selection, Edic-Trans shows a menu for several operations for the selected dictionary. Many operations are applicable for dictionaries. If the system displays all of them in a menu, a user may confuse the operation. Therefore, the menu system of Edic-Trans has 5 levels (0–4) according to the level of a user.

- *Level 0* is for users who only want to retrieve the dictionary. This is the default level just after invoking Edic-Trans system, as shown in figure 6.16, and has the following operations:
  - `SUPER`: An item to enter the dictionary selection mode.
  - `EXIT`: An item to quit from Edic-Trans.
  - `LEVEL`: An item to change menu level, which is discussed in this section.
  - `GET-MTF`: An operation to retrieve an MT format dictionary (a *neutral* dictionary.
  - `GET-DICT`: An operation to retrieve a Tree format dictionary, which is one of dictionaries for *processing*.[6]
  - `LIST-SESSION-LOG`: An operation to read a log file of Edic-Trans system. When a user runs `GET-MTF` or `GET-DICT`, the result of the retrieval is not only output to the terminal but written to a file (log file).[7] This command shows the contents of the log file.
  - `MTF-VSAM-TO-SUMMARY`: An operation to retrieve an MT format dictionary with various kind of restrictions. This is discussed in section 6.4.2.

```
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
1 :SUPER                                2 :EXIT
3 :LEVEL                                4 :DESCRIBE
5 :GET-MTF                              6 :GET-DICT
7 :LIST-SESSION-LOG                     8 :CLOSE-ALL-FILES
9 :MTF-VSAM-TO-SUMMARY                  10:DUMP-PS-TO-GET-DICT-PS
11:GET-DICT-PS-TO-GET-DICT-VSAM         12:DUMP-PS-TO-CALL-DIC-MAC-SRC-PS
13:CALL-DIC-MAC-SRC-PS-TO-GRD-SRC-PS    14:DUMP-PS-TO-GRD-SRC-PS
```

Figure 6.17: Edic-Trans Menu Level 1

- *Level 1* is for a grammar writer who needs to transform the dictionary format. In addition to the items of level 0, there are items related to the dictionary transformation as shown in figure 6.17. Details are discussed in later of this section. For example, the menu item

  `GET-DICT-PS-TO-GET-DICT-VSAM`

  is used to copy a Tree format dictionary (`GET-DICT`) entries into a dictionary file accessed by the translation system directly (`VSAM`).

- *Level 2* is for a more *sophisticated* grammar writer who wants to maintain the macro definition for a dictionary transformation.

- *Level 3* is for a lexicographer who wants to maintain the whole dictionary system as shown in figure 6.18. For example, the item

  `SOURCE-TO-MTF-VSAM`

  is an operation to copy a sequential file which contains MT format dictionary data to an MT format dictionary database which is implemented on a VSAM file. The sequential file is used to exchange the dictionary data among different organizations (Kyoto University, JICST, etc.) and to back-up the dictionary data.

[6]In the current system, there is no tool to retrieve a rule format dictionary discussed in chapter 5.
[7]This log file may be used as the input for the dictionary transformation.

```
SELECT TRANSFORMATION TYPE (LEVEL 3) FOR E-V-GEN:
1 :SUPER                                    2 :EXIT
3 :LEVEL                                    4 :DESCRIBE
5 :DESCRIBE-ALL                             6 :GET-MTF
7 :GET-DICT                                 8 :LIST-SESSION-LOG
9 :CLOSE-ALL-FILES                          10:SOURCE-TO-MTF-VSAM
11:LOG-PS-TO-MTF-VSAM                       12:MTF-VSAM-TO-SUMMARY
13:DUMP-PS-TO-SUMMARY                       14:MT-RETRIEVE-FROM-MTF-VSAM
15:MTF-VSAM-TO-DUMP-PS                      16:MTF-VSAM-TO-DUMP-PO
17:DUMP-PS-TO-GET-DICT-PS                   18:MTF-VSAM-TO-GET-DICT-PO
19:GET-DICT-PS-TO-GET-DICT-VSAM             20:DUMP-PS-TO-CALL-DIC-MAC-SRC-PS
21:CALL-DIC-MAC-SRC-PS-TO-GRD-SRC-PS        22:CALL-DIC-MAC-SRC-PO-TO-GRD-SRC-PO
23:DUMP-PS-TO-GRD-SRC-PS                    24:LOAD-WORD-TABLE-FOR-TRANS
25:LOAD-DFTSET-FOR-TRANS                    26:GRD-SRC-PS-TO-GRD-INT-PS
27:MTF-VSAM-TO-DUMP-PS-BY-KEY-LIST          28:KEY-LIST-SEQ-TO-KEY-LIST-VSAM
29:KEY-LIST-VSAM-TO-DUMP-PS                 30:EDIC-INTERN-SELECT
```

Figure 6.18: Edic-Trans Menu Level 3

- *Level 4* is for a programmer who needs to maintain the Edic-Trans system itself. There are several operations for internal data to execute the Edic-Trans system which is explained at the end of this section.

Dictionary Format Transformation

This section explains an example interaction to generate a dictionary entry for processing of analysis and generation from a *neutral* dictionary. To transform a dictionary format, a grammar writer or a lexicographer needs to execute four steps as follows.

Step 1: *Dictionary look up.* When a user wants to generate a dictionary entry for processing, he first needs to make a *copy* of the dictionary entry of a *neutral* dictionary (MTF dictionary).

Figure 6.19 shows this dictionary look up process. This operation is executed by the menu item

```
GET-MTF
```

and a user specifies the entry word. In the example in figure 6.19 and the rest of this section, a user is supposed to want to create a dictionary for 'affect.'

```
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
...
5 :GET-MTF                                  6 :GET-DICT
...
ENTER ITEM:5
ENTER KEY (EVALED) OR * FOR DEFAULT(increase).
ENTER>"AFFECT"

;;; KEY : "affect",
;;; VSAM: 'A51667.GEN.V.MTF.VSAM', VSAM-AREA-NUMBER: 153.
((SEQ 16)
 (E_LEX affect)
 (E_CAT V)
...
 (DOUBLE-END /-)
 (USAGE ((DERIVATION
           (E_DERIV_STATE_ACTION effect)
...
         (C-FRAME
            ((E_SURFACE_CASE1 SUBJ)
             (E_DEEP_CASE1 CPO)
             (E_OBLIGATORY1 1))
            ((E_SURFACE_CASE2 OBJ1)
             (E_DEEP_CASE2 OBJ)
             (E_SYNTACTIC_FORM2 1)
             (E_OBLIGATORY2 1))))))
```

Figure 6.19: Dictionary Look Up (**affect**)

Note that this operation GET-MTF not only outputs the content of the specified word, but writes it to a file whose name is EDIC.SESSION, which will be used for the dictionary transformation process.

Step 2: *Transformation from a MTF dictionary into a Macro Source format.* Next step of the dictionary transformation is to make a *macro source format* entry (dictionary expression using macro description) as discussed in section 5.6.

This operation is executed by the menu item

DUMP-PS-TO-CALL-DIC-MAC-SRC-PS (item 12 in this case[8])

[8]The item number for each operation is varied depend on dictionaries and levels of the menu.

as shown in figure 6.20. To execute this operation, Edic-Trans system asks a user for three parameters:

1. an input file name which contains MTF dictionary entries;
2. an output file which will contain macro source format entries;
3. a file name which contains the macro definition;

in this order. Edic-Trans prompts with the key words which explain what kind of parameters are requested and their default value. In this example, the file name has a *meaningful* default (`'A50650.TRAGEN.MACDEF'`), and others are dummy values.

After the execution of this process, a user gets the result of the transformation as shown in figure 6.21. Note that the command `X` of Edic-Trans is a kind of "escape command" to run an operating system command such as 'LIST.'

Step 3: *Macro Expansion.* When a user obtains a macro source format from an MTF dictionary, he runs the macro expansion operation as shown in figure 6.22. The menu item:

`CALL-DIC-MAC-SRC-PS-TO-GRD-SRC-PS` (item 13, in this example)

is used for this operation and asks a user for an input file name

`CALL-DIC-MAC-SRC-PS-FILE-NAME`

and an output file name

`CALL-DIC-GRD-SRC-PS-FILE-NAME.`

Note that the default of an input file name becomes the output file name in the previous operation (`AFFECT.MACSRC`), in contrast to the default of an output file name.

The result of macro expansion is shown in figure 6.23. This is a GRADE grammar rule (dictionary rule).

```
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
...
11:GET-DICT-PS-TO-GET-DICT-VSAM          12:DUMP-PS-TO-CALL-DIC-MAC-SRC-PS
13:CALL-DIC-MAC-SRC-PS-TO-GRD-SRC-PS     14:DUMP-PS-TO-GRD-SRC-PS
ENTER ITEM:12
DEFAULT FOR DUMP-PS-FILE-NAME IS:
 "'???####.WORK.MTFORMAT'"
ENTER VALUE (EVALED), * FOR DEFAULT OR ** FOR SKIP ALL:
VALUE:"EDIC.SESSION"
DEFAULT FOR CALL-DIC-MAC-SRC-PS-FILE-NAME IS:
 "'???####.WORK.CALLDIC'"
ENTER VALUE (EVALED), * FOR DEFAULT OR ** FOR SKIP ALL:
VALUE:"AFFECT.MACSRC"
DEFAULT FOR CALL-DIC-MACRO-DEF-PS-FILE-NAME IS:
 "'A50650.TRAGEN.MACDEF'"
ENTER VALUE (EVALED), * FOR DEFAULT OR ** FOR SKIP ALL:
VALUE:*
+EXECUTE TRANSFORMATION FOR E-V-GEN:
 FUNCTION: MTC-FUNCALL-TO-PS IN MTCBASIC
 ARGS    : ((WRITE-GENV-MACRO "'A51667.DICT.TEXT(VGMACRO)'")
            "EDIC.SESSION"
            "AFFECT.MACSRC"
            "'A50650.TRAGEN.MACDEF'")
 USING KINTERN
OK? (YES, NO OR SUB)YES
+VALUE IS:
 0
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
...
```

Figure 6.20: MT Format Dictionary to Macro Source Form

```
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
...
13:CALL-DIC-MAC-SRC-PS-TO-GRD-SRC-PS    14:DUMP-PS-TO-GRD-SRC-PS
ENTER ITEM:X LIST AFFECT.MACSRC NON
 A51388.AFFECT.MACSRC
INCLUDE('A50650.TRAGEN.MACDEF')

 RRINGV(affect,1)
   GENERV(affect,1,1,
     COMPGV(affect),
     CASEGV(SUBJ,CPO,,,,,1,OBJ1,OBJ,((1)),,,,1))

 END OF DATA
ENTER ITEM:13
```

Figure 6.21: Result of Transformation (Macro Source)

```
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
...
13:CALL-DIC-MAC-SRC-PS-TO-GRD-SRC-PS    14:DUMP-PS-TO-GRD-SRC-PS
ENTER ITEM:13
DEFAULT FOR CALL-DIC-MAC-SRC-PS-FILE-NAME IS:
 "AFFECT.MACSRC"
ENTER VALUE (EVALED), * FOR DEFAULT OR ** FOR SKIP ALL:
VALUE:*
DEFAULT FOR CALL-DIC-GRD-SRC-PS-FILE-NAME IS:
 "'???####.WORK.GRADE'"
ENTER VALUE (EVALED), * FOR DEFAULT OR ** FOR SKIP ALL:
VALUE:"AFFECT.GRDSRC"
+EXECUTE TRANSFORMATION FOR E-V-GEN:
 FUNCTION: UTI-MAC
 ARGS     : ("AFFECT.MACSRC" "AFFECT.GRDSRC")
 USING KINTERN
OK? (YES, NO OR SUB)YES
+VALUE IS:
 "AFFECT.GRDSRC"
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
...
```

Figure 6.22: Macro Source to GRADE Source

```
SELECT TRANSFORMATION TYPE (LEVEL 1) FOR E-V-GEN:
...
ENTER ITEM:X LIST INCREASE.GRDSRC NON
 A51388.AFFECT.GRDSRC
                  /**** affect ****/
affect_V_GENERATION.SG;
 SM;  ORDER(1); DETERMINISTIC;
 RIS;
  affect_V_GENER_1;
END_SG.affect_V_GENERATION;
   affect_V_GENER_1.RR;
 MI;  LEVEL(0,0); ORDER(1); TREE;
 MC;
       %((V CASE));
   CASE: %(X1 X2); X1,X2:DISORDER_SKIP( %( ? ) );
   X1.E_DEEP_CASE = 'CPO';
   X2.E_DEEP_CASE = 'OBJ';
   V.E_UID = '1';
  CR;
   **;
   X1.E_SURFACE_CASE <= 'SUBJ';
   X2.E_SURFACE_CASE <= 'OBJ1';
   X2.E_SYNTACTIC_FORM <= '((1))';
       V.E_MAIN_VERB <= 'affect';
  V.E_VERB_TYPE <= 'VERBONLY';
END_RR.affect_V_GENER_1;

 END OF DATA
```

Figure 6.23: Result of Macro Expansion

```
SELECT TRANSFORMATION TYPE (LEVEL 0) FOR E-N-GEN:
...
9 :IFG
ENTER ITEM:9
ENTER LEX OR Q FOR QUIT>"COMPUTER"
  コンピュータ (KONPYUUTA)
  計算機 (KEISAN-KI)
  電算機 (DEN-SAN-KI)
  電子計算機 (DENSHI-KEISAN-KI)
    ==> computer
ENTER LEX OR Q FOR QUIT>Q
```

Figure 6.24: Inverse Translation Dictionary Look Up

Step 4: *GRADE translation and Registration.* The final step of the dictionary format transformation is the GRADE translation and registration of the internal form into a grammar database. This phase is separate from the Edic-Trans system and a user uses the GRADE translator for standard (general) grammar rules.

6.4.2 Retrieval System

Edic-Trans system allows a user to retrieve dictionary data in three modes:

- *Simple dictionary look up.* As shown in figure 6.19, a user looks up the MTF dictionary with the menu item GET-MTF. A user also looks up a tree format dictionary with the menu item GET-DICT.

- *Inverse translation dictionary.* Usually one word has several translations. At the same time, one translated word has several counter parts. A lexicographer needs to check these words as shown in figure 6.24 with the menu item IFG. In this example, the English word **computer** has three translated words of Japanese words: KONPYUUTA, KEISAN-KI, DEN-SAN-KI, and DENSHI-KEISAN-KI.[9]

- *Edic-summary.* A lexicographer needs to get a summary from the dictionary. When a lexicographer wants to list the English words which have a semantic

[9]In the real system, romaji characters, such as KONPYUUTA, are not displaied.

marker (E_SEM) ON, that is, the words related to a *natural substances*, he can use the menu item

```
MTF-VSAM-SUMMARY
```

as shown in figure 6.25. The parameter

```
SUMMARY-INFO-PROPS
```

is the condition to the retrieval. In this case, the pattern `'((E_SEM ON))` specifies the condition: the words whose E_SEM's value is ON.

6.4.3 Implementation of Edic-Trans System

The Edic-Trans system guides a user who needs to know the information about the dictionaries. The information becomes complex, because the logical implementation of the dictionaries which are operated by a lexicographer is different from the physical implementation of the dictionaries. For example, the English analysis dictionary and the English generation dictionary share one MT format dictionary, that is, a *neutral* English dictionary. However, the operations for generating rule format dictionaries are different. On the other hand, the English analysis dictionary and the Japanese analysis dictionaries, for instance, share the information about the dictionary transformation. Therefore, the Edic-Trans system uses a hierarchical database to store the information about dictionary operations as shown in figure 6.26.

This hierarchical information is expressed with a dictionary definition such as shown in figure 6.27. The definition in this figure specifies the information about the English Verb generation dictionary (E-V-GEN), which is expressed as:

```
(DEFINE-DICTIONARY E-V-GEN
```

The definition of the dictionary based on the property-name and property-value pair. This dictionary inherits the information stored in the dictionaries:

E-GENERATION-DICTIONARY and E-V.

```
SELECT TRANSFORMATION TYPE (LEVEL 0) FOR E-N-GEN:
...
7 :CLOSE-ALL-FILES                    8 :MTF-VSAM-TO-SUMMARY
9 :IFG
ENTER ITEM:8
...
DEFAULT FOR SUMMARY-INFO-PROPS IS:
 NIL
ENTER VALUE (EVALED), * FOR DEFAULT OR ** FOR SKIP ALL:
VALUE:'((E_SEM OM))
...
OK? (YES, NO OR SUB)YES
;;; EDIC-PRINT-SUMMARY FOR 'A51667.GEN.N.MTF.VSAM' (87011013455256).
;;; AREA: 1, LEX: (E_LEX)
;;; INFO: ((E_SEM OM)).

; 404   : FET
;           E_SEM                 = ((OM))
; 541   : IC
;           E_SEM                 = ((OM))
; 633   : LED
;           E_SEM                 = ((OM))
; 2318  : board
;           E_SEM                 = ((OM))
; 3191  : component
;           E_SEM                 = ((OM))
...
; 6350  : glaze
;           E_SEM                 = ((OM AC))
; 6845  : hybrid circuit
;           E_SEM                 = ((OM OA))
...
; 14956 : wire
;           E_SEM                 = ((OM))

;;; EDIC SUMMARY (SUM)

+VALUE IS:
 NIL
```

Figure 6.25: Example of Edic-Summary

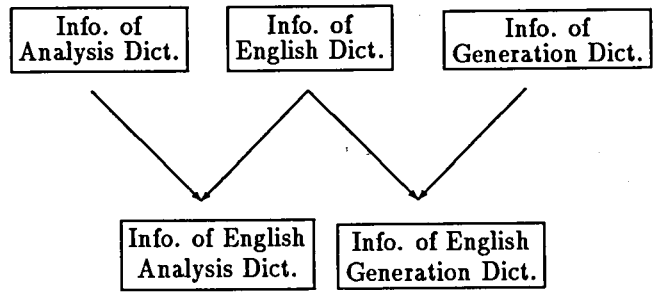


Figure 6.26: Hierarchy of Dictionary Information of Edic-Trans

```
(DEFINE-DICTIONARY E-V-GEN
  SUPER                  =  (E-GENERATION-DICTIONARY E-V)
  VSAM-AREA-NUMBER       =  1
  GET-DICT-VSAM-AREA-NUMBER = 153
  CALL-DIC-MAC-SRC-FUNCTION
      = (WRITE-GENV-MACRO "'A51667.DICT.TEXT(VGMACRO)'")
  GET-DICT-MAIN-FUNCTION
        = (MAKE-GENV-GETD "'A51667.DICT.TEXT(VSAM)'")
  )
```

Figure 6.27: Example of the Dictionary Definition of Edic-Trans

The property SUPER specifies this inheritance (hierarchy).

When the Edic-Trans system needs the function to generate a rule format dictionary (CALL-DIC) for this E-V-GEN, the system tries to get a property name

```
CALL-DIC-MAC-SRC-FUNCTION
```

In this case, the definition in figure 6.27 contains this property, as this information is specific to this English generation dictionary. Therefore, the Edic-Trans system uses it.

On the other hand, a file name of the MT format dictionary, for instance, is not defined in this expression, because it should be shared by English analysis and generation dictionaries. When the Edic-Trans system needs the file name, it searches the dictionary definition hierarchy defined by the property SUPER in left-to-right and depth-first order. Finally, Edic-Trans finds the file name of the English MT format dictionary in the definition of the English *neutral* dictionary.

This traversal of the dictionary definition hierarchy is one of the basic operations of the Edic-Trans system. In fact, Edic-Trans carries out the following operations of dictionary maintenance:

1. Reading an operation of a user.
2. Searching for the operation in the dictionary definition hierarchy.
3. Executing the operation which is expressed in the dictionary definition. If the operation requires additional information, such as a file name of the dictionary database, the system searching it again.

Because Edic Trans system simply executes these sequence, a new operation is easily added to the dictionary maintenance system by updating the dictionary definition.

6.5 Summary of this Chapter

A software environment is important for the development of a *practical* machine translation system. In this chapter, the software environment of the GRADE system is described. The GRADE system consists of the sub-systems:

1. The translator and the executor for GRADE grammar rules.

2. Tools for grammar development:

   - Tools for dynamic analysis of grammars: tree printer, tracer, breaker, disassembler, and interactive inspector.
   - Tools for static analysis of grammars: rule database retrieve programs, and corpus database retrieve programs.

3. Tools for dictionary maintenance: dictionary format transformation programs, dictionary retrieve programs, and menu system.

All of these sub-systems are implemented in UTILISP on a mainframe, and parts of the system are ported to the Symbolics 3600 series and other machines. These are used by grammar writers and lexicographers to develop the Mu machine translation system. This shows the usefulness of these tools.

However, there are still several problems. Grammar writers and lexicographers sometimes require an unexpected software tool for software system developers. Such tools appeared in the course of the development of the Mu machine translation system. The software tools should be designed to accept an unexpected demand by grammar writers and lexicographers. The framework of Edic-Trans discussed in section 6.4.1 is useful for such a requirement to augment the software tools.

Chapter 7

Conclusion

Software System for Machine Translation

In this thesis, a software system for a practical machine translation system is described. It has been developed for the Mu machine translation project, which was supported by the Science & Technology Agency of the Japanese Government from 1982 to 1986.

To develop a machine translation system, we need to study not only linguistic features of target sentences, which are described with grammar rules and dictionaries, but also a software system to support the development and maintenance of the grammar rules and dictionaries. For a *practical* machine translation, a software system is crucial, because we need to handle large and complex grammars and dictionaries.

In chapter 2, we discussed the requirement of a software system for a machine translation. A software system must have the following sub-systems:

1. A grammar writing language to describe complex linguistic phenomena.
2. A dictionary database to enable the handling of large word specific information.
3. Software packages for the development and maintenance of complex grammars and large dictionaries.

Grammar Writing Language

As a grammar writing language, the author developed the grammar writing language GRADE, which is described in chapter 3. GRADE has the following features to fulfil the requirement:

1. Rewriting rules are expressions in the form of *tree-to-tree transformation* with *annotations* for each node.

2. Rewriting rules have *powerful* capability to handle sophisticated linguistic phenomena.

3. A grammar can be divided into several parts as *subgrammars* (a set of rewriting rules), and each part is linked together as *subgrammar networks*.

4. A subgrammar can be written in the dictionary entries to express word specific linguistic phenomena.

5. *Para special nodes* are provided in trees for embedding ambiguities.

Each of them are presented in chapter 3.

To demonstrate the usefulness of the grammar writing language GRADE, we discussed typical problems in machine translation grammars and their solutions with the GRADE system in chapter 4:

- In the analysis of English sentences:
  1. Ambiguity of parts-of-speech.
  2. Complex constructions of sentences.
  3. Structural ambiguities.
- In the transfer from Japanese into English:
  1. Word selection.
  2. Adjustments of syntactic structures.

We also discussed the outline of the grammars developed in the Mu project. These showed that the grammar writing language GRADE is useful for the development of a practical machine translation system.

Dictionary Database System

Dictionaries are another key part of a machine translation system. To improve the usefulness of the dictionary database, the author proposed the dictionary database system, in which the dictionaries are separated into two parts:

- A *neutral* and *general purpose* dictionary.
- A dictionary for a processing which is transformed from the *neutral* dictionary.

In chapter 5, this dictionary database system and software tools to realize the separation are presented. This system has the following features:

1. We can develop and maintain the dictionary *independent* of the processing system.
2. We can use the same dictionary for different purposes, such as sentence analysis and sentence generation.

Software Tools for Development and Maintenance

Since grammars and dictionaries become very large and complex in a *practical* machine translation system, various software tools are necessary. As discussed in chapter 6, the author developed software packages for the development and maintenance of the grammars and dictionaries:

- Software for executing grammar rules.
- Tools for the development and maintenance of the grammar rules.
- Tools for the development and maintenance of the dictionaries.

These tools are used by grammar writers and lexicographers to develop the machine translation system in the Mu project. This proved the usefuleness of these tools.

168 項欠

Bibliography

[ALPAC66] ALPAC, Languages and machines: computers in translation and linguistics, National Academy of Sciences, National Research Council Publication 1416, Washington, DC (1966).

[Alshawi87] ALSHAWI, H., Processing Dictionary Definitions with Phrasal Pattern Hierarchies, *Computational Linguistics*, Vol. 13 (1987).

[Amano82] AMANO, S., HIRAKAWA, H., Parser for English-Japanese Machine Translation, IPSJ-WGNL, 32–1 (1982) (in Japanese).

[Boguraev87] BOGURAEV, B., Experiences with a Machine-Readable Dictionary, *Proc. of Third Annual Conf. of the UW Centre for the NOED*, pp. 37–50 (1987).

[Boitet79] BOITET, C., Automatic Production of CF and CS Analizers using A General Tree Transducer, Université Scientifique et Médicale de Grenoble (1979).

[Boitet82] BOITET, CH., ET AL, Implementation and Conversational Environment of ARIANE 78.4, *Proc. of COLING82*, pp. 19–28 (1982).

[Chikayama81] CHIKAYAMA, T., UtiLisp Manual, Technical Reports, Mathematical Engineering Section, Department of Mathematical Engineering and Instrumentation Physics, University of Tokyo (1981).

[Chikayama83] CHIKAYAMA, T., The development of UtiLisp System, *Transactions of IPSJ*, Vol. 24, No. 5, pp. 599–604 (1983) (in Japanese).

[Colmerauer70] COLMERAUER, A., LES SYSTEM-Q — Ou Un Formalisme pour Analyser et Synthetiser des Phrases sur Ordinatur, Université de Montreal (1970) (in French).

[Dymetman88] DYMETMAN, Y., ISABELLE, P., Reversible Logic Grammars for Machine Translation, *Proc. of 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, Center for MT, Carnegie Mellon University (1988).

[ETL78] ELECTRICAL TECHNICAL LABORATORY (ED.), Extended LINGOL — Programming System for Natural Language Processing, Inference Mechanizm Lab., ETL (1978) (in Japanese).

[ETL84] ELECTRICAL TECHNICAL LABORATORY, ET AL., Research on machine translation system (Japanese–English) of scientific and technological documents — Report on development of language processing system, ETL (1984) (in Japanese).

[Fujitsu83] FUJITSU, VSAM function manual, 78SP-1191-1, Fujitsu (1883) (in Japanese).

[Gazdar85] GAZDAR, G., PULLUM, E., SAG, I., Generalized Phrase Structure Grammar Harverd University Press, Cambridge, Massachusetts (1985).

[Hitaka82] HITAKA, T., YOSHIDA, S., INANAGA, A., Construction of Japanese Word Dictionary by Extended B-tree, IPSJ-WGNL, 33-8 (1982) (in Japanese).

[Iida82] IIDA, H., OGURA, K., NOMURA, H., English Noun Phrase Analysis Harmonizing ATN-Based Parser with Case Analysis, IPSJ-WGNL, 34-5 (1982) (in Japanese).

[Ikeda87] IKEDA, Y., TSUJII, J., NAGAO, M., Unification based grammar and its control in parsing, IEICEJ-WGNLC, 87-2 (1987) (in Japanese).

[Isahara84] ISAHARA, H., HOZUMI, T., Machine Translation based on Unified Method, *Proc. of Symposium on Natural Language Processing Technology*, IPSJ (1984) (in Japanese).

[Ishizaki86] ISHIZAKI, S., ISAHARA, H., Contextual Processing Technology, *Information Processing, IPSJ*, Vol. 27, No. 8, pp. 897–905 (1986) (in Japanese).

[JICST84] JICST, ET AL., Research on machine translation system (Japanese–English) of scientific and technological documents — Report on Development of Japanese-English scientific and technological documents dictionary database, JICST (1984) (in Japanese).

[JohnsonS75] JOHNSON, S. C., Yacc: Yet another Compiler-Compiler, Computing Science Technical Report No. 32, Bell Laboratories, NJ (1975).

[JohnsonR84] JOHNSON, R. L., KRAUWER, S., ROSNER, M. A., VARILE, G. B., The Design of the Kernel Architecture for the EUROTRA software, *Proc. of COLING84*, pp. 226–35 (1984).

[Katagiri85] KATAGIRI, H., NAKAMURA, J., TSUJII, J., NAGAO, M., Translation Experiment Envirnment of the Mu-Translation Project, IPSJ-WGNL, 47–9 (1985) (in Japanese).

[Knuth86] KNUTH, D. E., The TeXbook, Computers & Typesetting; A, Addison Wesley, Massachusetts (1986).

[Kogure84] KOGURE, K., YOKOO, A., SHIMAZU, A., NOMURA, H., Frame Editor for Dictionary Editing, IPSJ-WGNL, 45–1 (1984) (in Japanese).

[Kume87] KUME, M., TSUJII, J., NAGAO, M., The Construction of the Mu English-Japanese Translation System and the Translation Results, IPSJ-WGNL, 59–6 (1987) (in Japanese).

[Kusanagi82] KUSANAGI, Y., Problems on the Japanese Tense-Aspect Analysis, IPSJ-WGNL, 34–9 (1982) (in Japanese).

[Lamport86] LAMPORT, L, The LaTeX: A Document Preparation System Addison Wesley, Massachusetts (1986).

[Longman78] LONGMAN CO., Longman Dictionary of Contemporary English, Longman Co. (1978).

[Matsumoto82] MATSUMOTO, Y., TANAKA, H., Bottom-up parser in Prolog: BUP, IPSJ-WGNL, 34–6 (1982) (in Japanese).

[Matsumoto86] MATSUMOTO, Y., SUGIMURA, R., SAX: A Parsing System based on Logic Programming Languages, *Computer Software, JSSST*, Vol. 3, No. 4, pp. 4–11 (1986) (in Japanese).

[Melby83] MELBY, A. K., Computer-Assisted Translation Systems: The Standard Design and A Multi-level Design, *Proc. of Conference on Applied Natural Language Processing*, CA, pp. 174–7 (1983).

[Muraki83] MURAKI, K., ICHIYAMA, S., Natural Language Analysis in VENUS Machine Translation System, IPSJ-WGNL, 40–7 (1983) (in Japanese).

[Muraki84a] MURAKI, K., Dictionary Organization for Machine Translation, IPSJ-WGNL, 46–1 (1984) (in Japanese).

[Muraki84b] MURAKI, K., Japanese–English Machine Translation System with Knowledge Base and Language-independent Internal Structure, *Nikkei Electronics*, Nikkei, Tokyo, 1984–12–17, pp. 195–220 (1984) (in Japanese).

[Nagao76] NAGAO, M., TSUJII, J., PLATON — A New Programming Language for Natural Language Anaysis, *Journal of Computational Linguistics*, Microfiche 37 (1976).

[Nagao80a] NAGAO, M., TSUJII, J., MITAMURA, K., HIRAKAWA, H., KUME, M., A machine translation system from Japanese into English — another perspective of MT system, *Proc. of COLING80*, pp. 414–423 (1980).

[Nagao80b] NAGAO, M., TSUJII, J., UEDA, Y., TAKIYAMA, M., An attempt to computerized Dictionary Data Base, *Proc. of COLING80*, pp. 534–542 (1980).

[Nagao82a] NAGAO, M., TSUJII, J., YADA, K., An English Japanese Translation System of the Titles of Scientific and Enineering Papers, *Proc. of COLING82*, pp. 245–252 (1982).

[Nagao82b] NAGAO, M., NAKAMURA, J., A Parser Which Learns the Application Order of Rewriting Rules, *Proc. of COLING82*, pp. 253–258 (1982).

[Nagao82c] NAGAO, M., NAKAMURA, J., HATAZAKI, K., FUJITA, K., Application of Longman Dictionary for Machine Translation, IPSJ-WGNL, 29–5 (1982) (in Japanese).

[Nagao83a] NAGAO, M., Language Engineering, pp. 60–74, Aritificial Intelligence Series 2, Shoukou-dou, Tokyo (1983) (in Japanese).

[Nagao83b] NAGAO, M., Outline of a Machine Translation Project of the Japanese Government, IPSJ-WGNL, 38–2 (July 1983) (in Japanese).

[Nagao84] NAGAO, M., NISHIDA, T., TSUJII, J., Dealing with Incompleteness of Linguistic Knowledege on Language Translation — Transfer and Generation Stage of Mu Machine Translation Project, *Proc. of COLING84*, pp. 420–427 (1984).

[Nagao85a] NAGAO, M., TSUJII, J., NAKAMURA, J., The Japanese Government Project for Machine Translation, *Computational Linguistics*, Vol. 11, No. 2–3, pp. 91–110 (1985).

[Nagao85b] NAGAO, M., Evaluation of the Quality of Machine-Translated Sentences and the Control of Language, *Information Processing, IPSJ*, Vol. 26, No. 10, pp. 1197–1202 (1985) (in Japanese).

[Nagao85c] NAGAO, M., TSUJII., J., Lexical and Structural Transfer in a Machine Translation System, *Information Processing, IPSJ*, Vol. 26, No. 11, pp. 1261–70 (1985) (in Japanese).

[Nagao86a] NAGAO, M., TSUJII, J., The Transfer Phase of the Mu Machine Translation System, *Proc. of COLING86*, pp. 97–103 (1986).

[Nagao86b] NAGAO, M., TSUJII, J., NAKAMURA, J., Machine Translation from Japanese into English, *Proceedings of the IEEE*, Vol. 74, No. 7, pp. 993–1012 (1986).

[Nakamura83] NAKAMURA, J., Software System for Grammar Writing: GRADE, IPSJ-WGNL, 38–4 (1983) (in Japanese).

[Nakamura84a] NAKAURA, J., TSUJII, J., NAGAO, M., SAKAMOTO, Y., SATO, M., Dictionary Utilization Method in Mu-Project — Japanese–English Transfer Dictionary and English Generation Dictionary, *Proc. of Symposium on Natural Language Processing Technology*, IPSJ (1984) (in Japanese).

[Nakamura84b] NAKAMURA, J., NAGAO, M., Grammar Developing Tool for Machine Translation with Multi-Windows, *Proc. of 29th. Convention of IPSJ*, pp. 1225–6 (1984) (in Japanese).

[Nakamura84c] NAKAMURA, J., TSUJII, J., NAGAO, M., Grammar Writing System (GRADE) of Mu-Machine Translation Project and its Characteristics, *Proc. of COLING84*, pp. 338–343 (1984).

[Nakamura85a] NAKAMURA, J., TSUJII, J., NAGAO, M., Problems and Improvements of UtiLisp on Software System GRADE for Machine Translation, IPSJ-WGSYM, 35–2 (1985) (in Japanese).

[Nakamura85b] NAKAMURA, J., TSUJII, J., NAGAO, M., Grammar Writing System (GRADE) of Mu-Machine Translation Project and its Characteristics, *Journal of Information Processing, IPSJ*, Vol. 8, No. 2, pp. 93–100 (1985).

[Nakamura85c] NAKAMURA, J., Software Environment in Machine Translation, *Information Processing, IPSJ*, Vol. 26, No. 10, pp. 1184–1196 (1985) (in Japanese).

[Nakamura86a] NAKAMURA, J., FUJIGAKI, M., NAGAO, M., Extraction of Computer Processing Information from Longman Dictionary — Extraction of Hierarachy in Verb, *Proc. of 32th Convention of IPSJ*, pp. 1573–1574 (1986) (in Japanese).

[Nakamura86b] NAKAMURA, J., FUJIGAKI, M., NAGAO, M., Longman Dictionary Database and Extraction of Linguistic Information — Extraction of Hierarchy in Verb, SUURIKAGAKU-KOUKYUROKU, 593, RIMS, Kyoto University (1986) (in Japanese).

[Nakamura86c] NAKAMURA, J., TSUJII, J., NAGAO, M., Solutions for Problems of MT Parser — Methods used in Mu-Machine Translation Project —, *Proc. of COLING86*, pp. 133–135 (1986).

[Nakamura86d] NAKAMURA, J., TSUJII, J., NAGAO, M., A Utilization Method of Dictionary Databases in Machine Translation, *Transactions of IPSJ*, Vol. 27, No. 8, pp. 801–810 (1986) (in Japanese).

[Nakamura87a] NAKAMURA, J., Software Tools for Machine Translation, *Proc. of 1st Symposium of University Science*, pp. 169–184, Executive Commottee of 'Features of Japanese and Machine Translation' (1987) (in Japanese).

[Nakamura87b] NAKAMURA, J., SAKAI, K., NAGAO, M., Automatic Analysis of Semantical Relation between English Nouns by an Ordinal English Dictionary, IEICEJ-WGNLC, 86–23 (1987) (in Japanese).

[Nakamura87c] NAKAMURA, J., NAGAO, M., Automatic Extraction of Semantical Relation between English Nouns from Longman Dictionary, *Proc. of 34th Convention of IPSJ*, pp. 1291–1292 (1987) (in Japanese).

[Nakamura88] NAKAMURA, J., NAGAO, M., Extraction of Semantic Information from an Ordinary English Dictionary and its Evaluation, *Proc. of COLING88*, pp. 459–464 (1988).

[Nikkei83] NIKKEI ELECTRONICS (ED.), Trend of Japanese Machine Translation Systems aiming for a practical use, *Nikkei Electronics*, Nikkei, Tokyo, 1983–8–20, pp. 250–271, (1983) (in Japanese).

[NishidaF80] NISHIDA, F., TAKAMATSU, S., KUROKI, H., English-Japanese Translation through Case-Structure Conversion, *Proc. of COLING80*, pp. 447–454 (1980).

[NishidaF85] NISHIDA, F., TAKAMATSU, S., Intermediate Expression Transfer and Sentence Generation in Machine Translation, *Information Processing, IPSJ*, Vol. 26, No. 10, pp. 1165–1173 (1985) (in Japanese).

[NishidaT82] NISHIDA, T., DOSHITA, S., An English-Japanese Machine Translation System based on Formal Semantics of Natural Language, *Proc. of COLING82*, pp. 277–282 (1982).

[NishidaT83] NISHIDA, T., Studies on the Application of Formal Semantics to English-Japanese Machine Translation PhD., Dept. of Information Science, Kyoto Univ., (1983).

[Nitta82] NITTA, Y., OKAJIMA, A., YAMANO, F., ISHIHARA, K., A heuristic approach to English-into-Japanese machine translation, *Proc. of COLING82*, pp. 283–288 (1982)

[Pereira80] PEREIRA, F., WARREN, D., Definit Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Augumented Transition Networks, *Artificial Intelligence*, Vol. 13, pp. 231–78 (1980).

[Pratt73] PRATT, V. R., A Linguistics Oriented Programming Language, *Proc. of 3rd IJCAI*, pp. 372–81 (1973).

[Quezel78] QUEZEL-AMBRUNAZ, M., ARIANE-78: Systeme Interactif pour la Traduction Automatique Multilingue, Université Scientifique et Medicalé de Grenoble, (1987).

[Saito82] SAITO, Y., NOMURA, H., Functions of JMACS: Japanese Screen Oriented Editor, IPSJ-WGNL, 31–4 (1982) (in Japanese).

[Sakaki84] SAKAKI, H., HASHIMOTO, K., SUZUKI, M., NOGAITO, I., A Parsing Method Having Filtering Procedure, IPSJ-WGNL, 43–1 (1984)

[Sakamoto84] SAKAMOTO, Y., SATOH, M., ISHIKAWA, T, Lexicon Features for Japanese Syntactic Analysis in Mu-Project–JE, *Proc. of COLING84*, pp. 42–47 (1984).

[Sells85] SELLS, P., Lectures on Contemporary Syntactic Theories, CSLI Lecture Notes Number 3 CSLI, Standord University, Stanford (1985).

[Shieber84] SHIEBER, S. M., The Design of a Computer Language for Linguistic Information, *Proc. of COLING84*, pp. 362–6 (1984).

[Simmons72] SIMMONS, R. F., SLOCUM, J., Generating English Discourse from Semantic Networks, CACM, Vol. 15 pp. 891–905 (1972).

[Slocum84] SLOCUM, J., Machine Translation: its History, Current Status, and Future Prospects, *Proc. of COLING84*, pp. 546–561 (1984).

[Slype79] SLYPE, G. V., PIGOTT,I., Description de Systeme de Traduction Automatique SYSTRAN de la Communautes Europeenees, internal report, (1979).

[Tadenuma61] TADENUMA, Y, Research on Translation from Japanese into English with Computer (I), Electrical Technical Laboratory, No. 624 (1961) (in Japanese).

[Takamatsu85] TAKAMATSU, S., FUJITA, TANI, NISHIDA, F., Partial Transfer of Internal Expression and English Sentence Generation in Japanese–English Machine Translation, *Information Processing, IPSJ*, Vol. 26, No. 5, pp. 788–797 (1985) (in Japanese).

[Tamati85] TAMATI, T, A Historical Overview of Machine Translation, *Information Processing, IPSJ*, Vol. 26, No. 5, pp. 1140–1147 (1985) (in Japanese).

[Tsujii84] TSUJII, J., NAKAMURA, J., NAGAO, M., Analysis Grammar of Japanese in Mu-Project — A Procedural Approach to Analysis Grammar, *Proc. of COLING84*, pp. 267–274 (1984).

[Tsujii85a] TSUJII, J., Machine Translation, in Word Processor and Japanese Language Processing, extra issue, bit, Kyoritsu-Shuppan, Tokyo, pp. 254–267 (1985) (in Japanese).

[Tsujii85b] TSUJII, J., ET AL, Configuration of English-to-Japanese Translation System in Mu-project, IPSJ-WGNL, 50–3 (1985) (in Japanese).

[Tsujii86] TSUJII, J., Analysis Methods for a Natural Language Sentence, *Information Processing, IPSJ*, Vol. 27, No. 8, pp. 924–932 (1986) (in Japanese).

[Tsujii88] TSUJII, J., What is a cross-linguisticaly valid interpretation of discourse?, *Proc. of New Directions in Machine Translation*, John von Neuman Society for Computing Sciences, Budapest (1988).

[Tsurumaru84a] TSRUMARU, H., UCHIDA, A., HITAKA, T., YOSHIDA, S., The Extraction and Reorganization of Information from the Ordinary Japanese Language Dictionary (OJLD), IPSJ-WGNL, 43–6 (1984) (in Japanese).

[Tsurumaru86] Tsurumaru84b TSURUMARU, H., HITAKA, T., YOSHIDA, S., The Extraction of the Hierarchical Relation bewteen the Words from the Ordinary Japanese Laguage Dictionary (OJLD), IPSJ-WGNL, 45–4 (1984) (in Japanese).

[Uchida82] UCHIDA, H., ET AL., Japanese-English Machine Translation System: ATLAS/U, IPSJ-WGNL, 29–3 (1982) (in Japanese).

[Uchida84] UCHIDA, H., Multi-lingual Machine Translation System using Common Sence based on language-independent Conceptual Structure as Intermediate Representation, *Nikkei Electronics*, Nikkei, Tokyo, 1984–12–17, pp. 221–40 (1984) (in Japanese).

[Uchida85] UCHIDA, H., A Machine Translation System by Sentence Understanding — ATLAS/II, *Information Processing, IPSJ*, Vol. 26, No. 10, pp. 1217–1219 (1985) (in Japanese).

[Weinreb81] WEINREB, DANIEL, MOON, LISP Machine Manual, Fourth Edition, MIT Artificial Intelligence Lab., Cambrige, Massachusetts (1981).

[Woods70] WOODS, W. A., Transition Network Grammars for Natural Language Analysis, CACM, Vol. 13, pp. 591–606 (1970).

[Yada83] YADA, M., MAKOTO, M., Design for Utility System on Machine Translation, IPSJ-WGNL, 38–9 (1983) (in Japanese).

[Yamamoto86] YAMAMOTO, T., TSUJII, J., NAGAO, M., Disambiguation in the English-Japanese Machine Translation System of the Mu-Project, IPSJ-WGNL, 53–7 (1986) (in Japanese).

[Yamano85] YAMANO, F., OKAJIMA, J., OKAMOTO, R., Development Support Envirnment of ATHENE/E: English–Japanese Machine Translation System, *Proc. of 30th, Convention of IPSJ*, pp. 1577–8 (1985) (in Japanese).

[Yoshida85] YOSHIDA, S., HITAKA, T., Parsing Methods for Machine Translation, *Information Processing, IPSJ*, Vol. 26, No. 10, pp. 1157–1164 (1985) (in Japanese).

[Yoshida86] YOSHIDA, S., Problems in Dictionary Construction, *Information Processing, IPSJ*, Vol. 27, No. 8, pp. 933–939 (1986) (in Japanese).

[Yuasa85] YUASA, T., HAGIYA, M., Kyoto Common Lisp Report, Teikoku Insatsu, Tokyo, Japan (1985).

Publications

Main Publications

[1] NAGAO, M., NAKAMURA, J., A Parser Which Learns the Application Order of Rewriting Rules, *Proc. of COLING82*, pp. 253–258 (1982).

[2] NAKAMURA, J., TSUJII, J., NAGAO, M., Grammar Writing System (GRADE) of Mu-Machine Translation Project and its Characteristics, *Proc. of COLING84*, pp. 338–343 (1984).

[3] TSUJII, J., NAKAMURA, J., NAGAO, M., Analysis Grammar of Japanese in Mu-Project — A Procedural Approach to Analysis Grammar, *Proc. of COLING84*, pp. 267–274 (1984).

[4] NAKAMURA, J., TSUJII, J., NAGAO, M., Grammar Writing System (GRADE) of Mu-Machine Translation Project and its Characteristics, *Journal of Information Processing, IPSJ*, Vol. 8, No. 2, pp. 93–100 (1985).

[5] NAKAMURA, J., Software Environment in Machine Translation, *Information Processing, IPSJ*, Vol. 26, No. 10, pp. 1184–1196 (1985) (in Japanese).

[6] NAGAO, M., TSUJII, J., NAKAMURA, J., SAKAMOTO, Y., TORIUMI, T., SATO, M., Outline of Machine Translation Project of the Science and Technology Agency, *Information Processing, IPSJ*, Vol. 26, No. 10, pp. 1203–1213 (1985) (in Japanese).

[7] NAGAO, M., TSUJII, J., NAKAMURA, J., The Japanese Government Project for Machine Translation, *Computational Linguistics*, Vol. 11, No. 2–3, pp. 91–110 (1985).

[8] NAGAO, M., TSUJII, J., NAKAMURA, J., Science and Technology Agency's Mu Machine Translation Project, *Future Generation Computer System*, Vol. 2, pp. 125-139, North-Holland (1986).

[9] NAGAO, M., TSUJII, J., NAKAMURA, J., Machine Translation from Japanese into English, *Proceedings of the IEEE*, Vol. 74, No. 7, pp. 993–1012 (1986).

[10] NAKAMURA, J., TSUJII, J., NAGAO, M., Solutions for Problems of MT Parser — Methods used in Mu-Machine Translation Project —, *Proc. of COLING86*, pp. 133–135 (1986).

[11] NAKAMURA, J., TSUJII, J., NAGAO, M., A Utilization Method of Dictionary Databases in Machine Translation, *Transactions of IPSJ*, Vol. 27, No. 8, pp. 801–810 (1986) (in Japanese).

[12] NAKAMURA, J., TSUJII, J., NAGAO, M., GRADE: A Software Environment for Machine Translation, *Computers and Translation*, 3, pp. 69–82, Kluwer Academic Publishers (1988).

[13] NAKAMURA, J., NAGAO, M., Extraction of Semantic Information from an Ordinary English Dictionary and its Evaluation, *Proc. of COLING88*, pp. (1988).

Other Publications

[14] NAGAO, M., NAKAMURA, J., HATAZAKI, K., FUJITA, K., Application of Longman Dictionary for Machine Translation, IPSJ-WGNL, 29–5 (1982) (in Japanese).

[15] NAGAO, M., TSUJII, J., NAKAMURA, J., Report on COLING84, IPSJ-WGNL, 45–5 (1984) (in Japanese).

[16] NAKAMURA, J., Software System for Grammar Writing: GRADE, IPSJ-WGNL, 38–4 (1983) (in Japanese).

[17] NAKAURA, J., TSUJII, J., NAGAO, M., SAKAMOTO, Y., SATO, M., Dictionary Utilization Method in Mu-Project — Japanese–English Transfer Dictionary and English Generation Dictionary, Proc. of symposium on Natural Language Processing Technology, IPSJ (1984) (in Japanese).

[18] NAKAMURA, J., NAGAO, M., Grammar Developing Tool for Machine Translation with Multi-Windows, *Proc. of 29th. Convention of IPSJ*, pp. 1225–6 (1984) (in Japanese).

[19] NAKAMURA, J., TSUJII, J., NAGAO, M., Grammar Writing Language: GRADE, *Proc. of 1st Convention of JSSST*, pp. 243–246 (1984) (in Japanese).

[20] KATAGIRI, H., NAKAMURA, J., TSUJII, J., NAGAO, M., Translation Experiment Envirnment of the Mu-Translation Project, IPSJ-WGNL, 47–9 (1985) (in Japanese).

[21] NAKAMURA, J., TSUJII, J., NAGAO, M., Problems and Improvements of UtiLisp on Software System GRADE for Machine Translation, IPSJ-WGSYM, 35–2 (1985) (in Japanese).

[22] NAKAMURA, J., FUJIGAKI, M., NAGAO, M., Extraction of Computer Processing Information from Longman Dictionary — Extraction of Hierarachy in Verb, *Proc. of 32th Convention of IPSJ*, pp. 1573–1574 (1986) (in Japanese).

[23] NAKAMURA, J., FUJIGAKI, M., NAGAO, M., Longman Dictionary Database and Extraction of Linguistic Information — Extraction of Hierarchy in Verb, SUURIKAGAKU-KOUKYUROKU, 593, RIMS, Kyoto University (1986) (in Japanese).

[24] NAKAMURA, J., Natural Language Processing and its Software System, IEICEJ-WGNLC, 86–5 (1986) (in Japanese).

[25] NAKAMURA, J., Software Tools for Machine Translation, *Proc. of 1st Symposium of University Science*, pp. 169–184, Executive Commottee of 'Features of Japanese and Machine Translation' (1987) (in Japanese).

[26] NAKAMURA, J., SAKAI, K., NAGAO, M., Automatic Analysis of Semantical Relation between English Nouns by an Ordinal English Dictionary, IEICEJ-WGNLC, 86–23 (1987) (in Japanese).

[27] NAKAMURA, J., NAGAO, M., Automatic Extraction of Semantical Relation between English Nouns from Longman Dictionary, *Proc. of 34th Convention of IPSJ*, pp. 1291–1292 (1987) (in Japanese).

Abbreviations

IPSJ: Information Processing Society of Japan

WGNL: Natural Language

WGSYM: Symbol Manipulation

IEICEJ: The Institute of Electronics, Information and Communication Engineers of Japan

WGNLC: Natural Language Processing and Models of Communication

JSSST: Japan Society for Software Science and Technology

COLING: International Conference on Computational Linguistics

IJCAI: Internatinal Joint Conference on Artificial Intelligence